

Operator Precedence ω -languages

Federica Panella¹, Matteo Pradella¹, Violetta Lonati², Dino Mandrioli¹

¹ DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
 {panella, pradella, mandrioli}@elet.polimi.it

² DI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
 lonati@di.unimi.it

Abstract. ω -languages are becoming more and more relevant nowadays when most applications are “ever-running”. Recent literature, mainly under the motivation of widening the application of model checking techniques, extended the analysis of these languages from the simple regular ones to various classes of languages with “visible syntax structure”, such as visibly pushdown languages (VPLs). Operator precedence languages (OPLs), instead, were originally defined to support deterministic parsing and, though seemingly unrelated, exhibit interesting relations with these classes of languages: OPLs strictly include VPLs, enjoy all relevant closure properties and have been characterized by a suitable automata family and a logic notation.

In this paper we introduce operator precedence ω -languages (ω OPLs), investigating various acceptance criteria and their closure properties. Whereas some properties are natural extensions of those holding for regular languages, others required novel investigation techniques. Application-oriented examples show the gain in expressiveness and verifiability offered by ω OPLs w.r.t. smaller classes.

Keywords: ω -languages, Operator precedence languages, Push-down automata, Closure properties, Infinite-state model checking.

1 Introduction

Languages of infinite strings, i.e. ω -languages, have been introduced to model nonterminating processes; thus they are becoming more and more relevant nowadays when most applications are “ever-running”, often in a distributed environment. The pioneering work by Büchi and others investigated their main algebraic properties in the context of finite state machines, pointing out commonalities and differences w.r.t. the finite length counterpart [4,16].

More recent literature, mainly under the motivation of widening the application of model checking techniques to language classes as wide as possible, extended this analysis to various classes of languages with “visible structure”, i.e., languages whose syntax structure is immediately visible in their strings: parenthesis languages, tree languages, visibly pushdown languages (VPLs) [1] are examples of such classes.

Operator precedence languages, instead, were defined by Floyd in the 1960s with the original motivation of supporting deterministic parsing, which is trivial for visible structure languages but is crucial for general context-free languages such as programming languages [7], where structure is often left implicit (e.g. in arithmetic expressions). Recently, these seemingly unrelated classes of languages have been shown to

share most major features; precisely OPLs strictly include VPLs and enjoy all the same closure properties [6]. This observation motivated characterizing OPLs in terms of a suitable automata family [10] and in terms of a logic notation [11], which was missing in previous literature.

In this paper we further the investigation of OPLs properties to the case of infinite strings, i.e., we introduce and study operator precedence ω -languages (ω OPLs). As for other families, we consider various acceptance criteria, their mutual expressiveness relations, and their closure properties. Not surprisingly, some properties are natural extensions of those holding for, say, regular languages or VPLs, whereas others required different and novel investigation techniques essentially due to the more general managing of the stack. These closures and the decidability of the emptiness problem are a necessary step towards the possibility of performing infinite-state model checking. Simple application-oriented examples show the considerable gain in expressiveness and verifiability offered by ω OPLs w.r.t. previous classes.

The paper is organized as follows. The next section provides basic concepts on operator precedence languages of finite-length words and on operator precedence automata able to recognize them. Section 3 defines operator precedence automata which can deal with infinite strings, analyzing various classical acceptance conditions for ω -abstract machines. Section 4 proves the closure properties they enjoy w.r.t typical operations on ω -languages and shows also that the emptiness problem is decidable for these formalisms. Finally, Section 5 draws some conclusions.

2 Preliminaries

Operator precedence languages [6,7] have been characterized in terms of both a generative formalism (operator precedence grammars, OPGs) and an equivalent operational one (operator precedence automata, OPAs, named Floyd automata or FAs in [10]), but in this paper we consider the latter, as it is better suited to model and verify nonterminating computations of systems. We first recall the basic notation and definition of operator precedence automata able to recognize words of finite length, as presented in [10].

Let Σ be an alphabet. The empty string is denoted ε . Between the symbols of the alphabet three types of operator precedence (OP) binary relations can hold: *yields* precedence, *equal* in precedence and *takes* precedence, denoted $<$, \doteq and $>$ respectively. We use a special symbol $\#$ not in Σ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the lookback and lookahead of one character to determine the next action to perform [8]. The initial $\#$ can only yield precedence, and other symbols can only take precedence on the ending $\#$.

Definition 1. An operator precedence matrix (OPM) M over an alphabet Σ is a $|\Sigma \cup \{\#\}| \times |\Sigma \cup \{\#\}|$ array that with each ordered pair (a, b) associates the set M_{ab} of OP relations holding between a and b . M is conflict-free iff $\forall a, b \in \Sigma, |M_{ab}| \leq 1$. We call (Σ, M) an operator precedence alphabet if M is a conflict-free OPM on Σ .

Between two OPMs M_1 and M_2 , we define set inclusion and union:

$$M_1 \subseteq M_2 \text{ if } \forall a, b : (M_1)_{ab} \subseteq (M_2)_{ab}, \quad M = M_1 \cup M_2 \text{ if } \forall a, b : M_{ab} = (M_1)_{ab} \cup (M_2)_{ab}$$

If $M_{ab} = \{\circ\}$, with $\circ \in \{<, \doteq, >\}$, we write $a \circ b$. For $u, v \in \Sigma^*$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$. Two matrices are *compatible* if their union is conflict-free. A matrix is *complete* if it contains no empty case.

In the following we assume that M is \doteq -acyclic, which means that $c_1 \doteq c_2 \doteq \dots \doteq c_k \doteq c_1$ does not hold for any $c_1, c_2, \dots, c_k \in \Sigma, k \geq 1$.

Definition 2. A nondeterministic operator precedence automaton (OPA) is a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ where:

- (Σ, M) is a precedence alphabet,
- Q is a set of states (disjoint from Σ),
- $I \subseteq Q$ is a set of initial states,
- $F \subseteq Q$ is a set of final states,
- $\delta : Q \times (\Sigma \cup Q) \rightarrow 2^Q$ is the transition function.

The transition function can be seen as the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \rightarrow 2^Q \quad \delta_{\text{flush}} : Q \times Q \rightarrow 2^Q$$

An OPA can be represented by a graph with Q as the set of vertices and $\Sigma \cup Q$ as the set of edge labels: there is an edge from state q to state p labeled by $a \in \Sigma$ if and only if $p \in \delta_{\text{push}}(q, a)$ and there is an edge from state q to state p labeled by $r \in Q$ if and only if $p \in \delta_{\text{flush}}(q, r)$. To distinguish flush transitions from push transitions we denote the former ones by a double arrow.

To define the semantics of the automaton, we introduce some notation. We use letters p, q, p_i, q_i, \dots for states in Q and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked symbols*.

Let Γ be $(\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in Γ as $[a \ q]$, $[a' \ q]$, or $[\# \ q]$, respectively. We set $\text{symbol}([a \ q]) = \text{symbol}([a' \ q]) = a$, $\text{symbol}([\# \ q]) = \#$, and $\text{state}([a \ q]) = \text{state}([a' \ q]) = \text{state}([\# \ q]) = q$. Given a string $\beta = B_1 B_2 \dots B_n$ with $B_i \in \Gamma$, we set $\text{state}(\beta) = \text{state}(B_n)$.

A *configuration* is any pair $C = \langle \beta, w \rangle$, where $\beta = B_1 B_2 \dots B_n \in \Gamma^*$, $\text{symbol}(B_1) = \#$, and $w = a_1 a_2 \dots a_m \in \Sigma^* \#$. A configuration represents both the contents β of the stack and the part of input w still to process.

A computation (run) of the automaton is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $\text{symbol}(B_n)$ and a_1 :

push move: if $\text{symbol}(B_n) \doteq a_1$ then $C_1 = \langle \beta[a_1 \ q], a_2 \dots a_m \rangle$, with $q \in \delta_{\text{push}}(\text{state}(\beta), a_1)$;

mark move: if $\text{symbol}(B_n) < a_1$ then $C_1 = \langle \beta[a_1' \ q], a_2 \dots a_m \rangle$, with $q \in \delta_{\text{push}}(\text{state}(\beta), a_1)$;

flush move: if $\text{symbol}(B_n) > a_1$ then let i the greatest index such that $\text{symbol}(B_i) \in \Sigma'$ (such index always exists). Then $C_1 = \langle B_1 B_2 \dots B_{i-2}[\text{symbol}(B_{i-1}) \ q], a_1 a_2 \dots a_m \rangle$, with $q \in \delta_{\text{flush}}(\text{state}(B_n), \text{state}(B_{i-1}))$.

Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push} ; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that

delimit the portion of the stack to be removed; notice that in this move the input symbol is not consumed and it remains available for the following move.

Finally, we say that a configuration $[\# q_I]$ is *starting* if $q_I \in I$ and a configuration $[\# q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(A) = \left\{ x \mid \langle [\# q_I], x\# \rangle \vdash^* \langle [\# q_F], \# \rangle, q_I \in I, q_F \in F \right\}.$$

Remark 1. The assumption on the \doteq -acyclicity has been introduced in previous literature [6,10] to prevent the construction of operator precedence grammars with unbounded length of production's right hand sides (r.h.s.). Correspondingly, in presence of \doteq -cycles of an OPM, an OPA could be compelled to an unbounded growth of the stack before applying a flush move. The \doteq -acyclicity hypothesis could be replaced by the weaker restriction of production's r.h.s. of bounded length in grammars and a bounded number of consecutive push moves in automata, or could be removed at all by allowing such unbounded forms of grammars – e.g. with regular expressions as r.h.s. – and automata. In this paper we accept a minimal loss of generation³ power and assume the simplifying assumption of \doteq -acyclicity.

An OPA is *deterministic* when I is a singleton and $\delta_{\text{push}}(q, a)$ and $\delta_{\text{flush}}(q, p)$ have at most one element, for every $q, p \in Q$ and $a \in \Sigma$.

An *operator precedence transducer* can be defined in the usual way as a tuple $\mathcal{T} = \langle \Sigma, M, Q, I, F, O, \delta, \eta \rangle$ where Σ, M, Q, I, F are defined as in Definition 2, O is a finite set of output symbols, the transition function δ and the output function η are defined by $\langle \delta, \eta \rangle : Q \times (\Sigma \cup Q) \rightarrow \mathcal{P}_F(Q \times O^*)$, where \mathcal{P}_F denotes the set of finite subsets of $(Q \times O^*)$, and $\langle \delta, \eta \rangle$ can be seen as the union of two disjoint functions, $\langle \delta_{\text{push}}, \eta_{\text{push}} \rangle : Q \times \Sigma \rightarrow \mathcal{P}_F(Q \times O^*)$ and $\langle \delta_{\text{flush}}, \eta_{\text{flush}} \rangle : Q \times Q \rightarrow \mathcal{P}_F(Q \times O^*)$.

A *configuration* of the transducer is denoted $\langle \beta, w \rangle \downarrow z$, where $C = \langle \beta, w \rangle$ is the configuration of the underlying OPA and the string after \downarrow represents the output of the automaton in the configuration. The transition relation \vdash is naturally extended from OPAs, concatenating the output symbol produced at each move with those generated in the previous moves. The *transduction* $\tau : I^* \rightarrow \mathcal{P}_F(O^*)$ generated by \mathcal{T} is defined by

$$\tau(x) = \left\{ z \mid \langle [\# q_I], x\# \rangle \downarrow \varepsilon \vdash^* \langle [\# q_F], \# \rangle \downarrow z, q_I \in I, q_F \in F \right\}$$

Example 1. As an introductory example, consider a language of queries on a database expressed in relational algebra. We consider a subset of classical operators (union, intersection, selection σ , projection π and natural join \bowtie). Just like mathematical operators, the relational operators have precedences between them: unary operators σ and π have highest priority, next highest is the “multiplicative” operator \bowtie , lowest are the “additive” operators \cup and \cap .

Denote as T the set of tables of the database and, for the sake of simplicity, let E be a set of conditions for the unary operators. The OPA depicted in Figure 1 accepts the language of queries without parentheses on the alphabet $\Sigma = T \cup \{\bowtie, \cup, \cap\} \cup \{\sigma, \pi\} \times E$,

³ An example language that cannot be generated with an \doteq -acyclic OPM is the following: $\mathcal{L} = \{a^n(bc)^n \mid n \geq 0\} \cup \{b^n(ca)^n \mid n \geq 0\} \cup \{c^n(ab)^n \mid n \geq 0\}$

where we use letters $A, B, R \dots$ for elements in T and we write σ_{expr} for a pair (σ, expr) of selection with condition expr (similarly for projection π_{expr}). The same figure also shows an accepting computation on input $A \cup B \bowtie C \bowtie \pi_{\text{expr}} D$.

Notice that the sentences of this language show the same structure as arithmetic expressions with prioritized operators and without parentheses, which cannot be represented by VPAs due to the particular shape of their OPM [6].

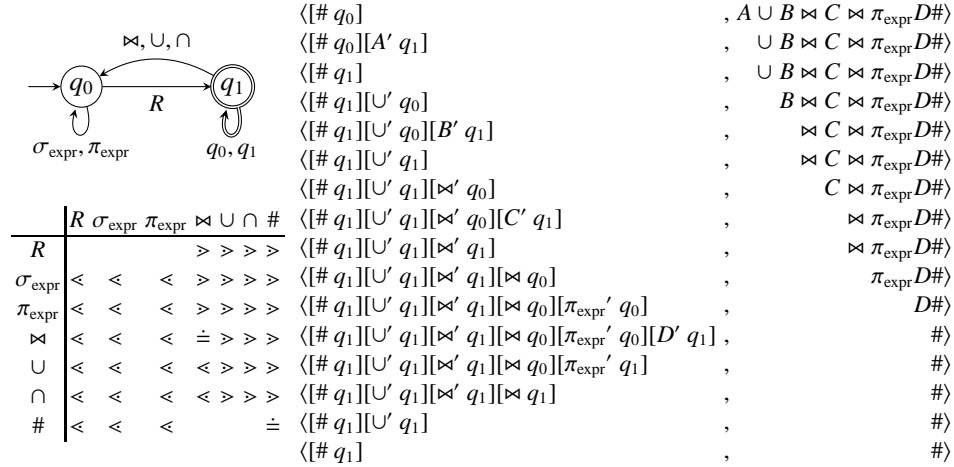


Fig. 1: Automaton, precedence matrix and example of computation for language of Example 1.

Let (Σ, M) be a precedence alphabet.

Definition 3. A simple chain is a word $a_0 a_1 a_2 \dots a_n a_{n+1}$, written as $\langle^{a_0} a_1 a_2 \dots a_n^{a_{n+1}} \rangle$, such that: $a_0 \in \Sigma \cup \{\#\}$, $a_i \in \Sigma$ for every $i : 1 \leq i \leq n+1$, $M_{a_0 a_{n+1}} \neq \emptyset$, and $a_0 \triangleleft a_1 \doteq a_2 \dots a_{n-1} \doteq a_n \triangleright a_{n+1}$.

A composed chain is a word $a_0 x_0 a_1 x_1 a_2 \dots a_n x_n a_{n+1}$, where $\langle^{a_0} a_1 a_2 \dots a_n^{a_{n+1}} \rangle$ is a simple chain, and $x_i \in \Sigma^*$ is the empty word or is such that $\langle^{a_i} x_i^{a_{i+1}} \rangle$ is a chain (simple or composed), for every $i : 0 \leq i \leq n$. Such a composed chain will be written as $\langle^{a_0} x_0 a_1 x_1 a_2 \dots a_n x_n a_{n+1} \rangle$.

A word w over (Σ, M) is compatible with M iff for each pair of consecutive letters c, d in w it holds that $M_{cd} \neq \emptyset$, and for each factor x of $\# w \#$ such that $x = a_0 x_0 a_1 x_1 a_2 \dots a_n x_n a_{n+1}$ where $a_0 \triangleleft a_1 \doteq a_2 \dots a_{n-1} \doteq a_n \triangleright a_{n+1}$ and $x_i \in \Sigma^*$ is the empty word or is such that $\langle^{a_i} x_i^{a_{i+1}} \rangle$ is a chain (simple or composed) for every $0 \leq i \leq n$, it holds that $M_{a_0 a_{n+1}} \neq \emptyset$.

Definition 4. Let \mathcal{A} be an operator precedence automaton. A support for the simple chain $\langle^{a_0} a_1 a_2 \dots a_n^{a_{n+1}} \rangle$ is any path in \mathcal{A} of the form

$$\xrightarrow{a_0} q_0 \xrightarrow{a_1} q_1 \longrightarrow \dots \longrightarrow q_{n-1} \xrightarrow{a_n} q_n \xrightarrow{q_0} q_{n+1} \quad (1)$$

Notice that the label of the last (and only) flush is exactly q_0 , i.e. the first state of the path; this flush is executed because of relation $a_n \triangleright a_{n+1}$.

A support for the composed chain $\langle^{a_0} x_0 a_1 x_1 a_2 \dots a_n x_n^{a_{n+1}} \rangle$ is any path in \mathcal{A} of the form

$$\xrightarrow{a_0} q_0 \xrightarrow{x_0} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{x_1} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \xrightarrow{x_n} q'_n \xrightarrow{q'_0} q_{n+1} \quad (2)$$

where, for every $i : 0 \leq i \leq n$:

- if $x_i \neq \varepsilon$, then $\xrightarrow{a_i} q_i \xrightarrow{x_i} q'_i$ is a support for the chain $\langle^{a_i} x_i^{a_{i+1}} \rangle$, i.e., it can be decomposed as $\xrightarrow{a_i} q_i \xrightarrow{x_i} q''_i \xrightarrow{q_i} q'_i$.
- if $x_i = \varepsilon$, then $q'_i = q_i$.

Notice that the label of the last flush is exactly q'_0 .

The chains fully determine the structure of the parsing of any automaton on a word compatible with M , and hence the structure of the syntax tree of the word. Indeed, if the automaton performs the computation $\langle [a \ q_0] , \ xb \rangle \vdash^* \langle [a \ q] , \ b \rangle$ on a factor axb , then $\langle^a x^b \rangle$ is necessarily a chain over (Σ, M) and there exists a support like (2) with $x = x_0 a_1 \dots a_n x_n$ and $q_{n+1} = q$.

3 Operator precedence ω -languages and automata

Let us now generalize operator precedence automata to deal with words of infinite length and to model nonterminating computations.

Traditionally, ω -automata have been classified on the basis of the acceptance condition of infinite words they are equipped with. All acceptance conditions refer to the occurrence of states which are visited in a computation of the automaton, and they generally impose constraints on those states that are encountered infinitely (or also finitely) often during a run. Classical notions of acceptance (introduced by Büchi [4], Muller [12], Rabin [14], Streett [15]) can be naturally adapted to ω -automata for operator precedence languages and can be characterized according to a peculiar acceptance component of the automaton on ω -words. We first introduce the model of nondeterministic Büchi-operator precedence ω -automata with acceptance by final state; other models are presented in Section 3.3.

As usual, we denote by Σ^ω the set of infinite-length words over Σ . Thus, the symbol $\#$ occurs only at the beginning of an ω -word. Given a precedence alphabet (Σ, M) , the definition of an ω -word compatible with the OPM M and the notion of syntax tree of an infinite-length word are the natural extension of these concepts for finite strings.

Definition 5. A nondeterministic Büchi-operator precedence ω -automaton (ω OPBA) is given by a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, where Σ, Q, I, F, δ are defined as for OPAs; the operator precedence matrix M is restricted to be a $|\Sigma \cup \{\#\}| \times |\Sigma|$ array, since ω -words are not terminated by the delimiter $\#$.

Configurations and (*infinite*) *runs* are defined as for operator precedence automata on finite-length words. Then, let “ $\exists^\omega i$ ” be a shorthand for “there exist infinitely many i ” and let \mathcal{S} be a run of the automaton on a given word $x \in \Sigma^\omega$. Define $In(\mathcal{S}) = \{q \in Q \mid \exists^\omega i \langle \beta_i, x_i \rangle \in \mathcal{S} \text{ with } state(\beta_i) = q\}$ as the set of states that occur infinitely often at the

top of the stack of configurations in S . A run S of an ω OPBA on an infinite word $x \in \Sigma^\omega$ is *successful* iff there exists a state $q_f \in F$ such that $q_f \in \text{In}(S)$. \mathcal{A} *accepts* $x \in \Sigma^\omega$ iff there is a successful run of \mathcal{A} on x . Furthermore, let the ω -language *recognized* by \mathcal{A} be $L(\mathcal{A}) = \{x \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } x\}$.

Operator precedence ω -transducers are defined in the natural way as for finite-length words.

3.1 Some examples

Example 2. Consider a software system which is supposed to work forever and may serve interrupt requests issued by different users. The system can manage three types of interrupts with different levels of priority, that affect the order by which they are served by the system: pending lower priority interrupts are postponed in favor of higher priority ones.

This policy can be naturally specified by defining an alphabet of letters for ordinary procedures and for interrupt symbols, and by formalizing the priority level among the interrupt requests as OP relationships in the precedence matrix of an operator precedence automaton on infinite-length words: an interrupt yields precedence (\leq) to higher priority ones, which will be handled first, and takes precedence (\geq) on lower priority requests, whose processing is then suspended. Figure 2 shows an ω OPBA with acceptance condition by final state which models the behavior of a system which may execute two functions denoted a and b , that may be suspended by interrupts of types $\text{int}_0, \text{int}_1$ and int_2 with increasing level of priority. Calls and returns of the procedures are denoted $\text{call}_a, \text{call}_b, \text{ret}_a, \text{ret}_b$. A request is actually served as soon as the corresponding interrupt symbol is flushed from the top of the stack. Figure 2 also presents the precedence matrix and an example computation of the system for the infinite string $\text{call}_a \text{call}_b \text{ret}_b \text{call}_b \text{int}_1 \text{int}_2 \text{int}_0 \text{ret}_b \dots$.

Several variations of the above policy can be specified as well by similar ω OPBAs; e.g., we might wish to formalize that high priority interrupts flush pending calls, whereas lower priority ones let the system resume serving pending calls once the interrupt has been served. We might also introduce an explicit symbol to formalize the end of serving an interrupt and specify that some events are disabled while serving interrupts with a given priority, etc.

Example 3. Operator precedence automata on infinite-length words can also be used to model the run-time behavior of database systems, e.g., for modeling sequences of users' transactions with possible rollbacks. Other systems that exhibit an analogous behavior are revision control (or *versioning*) systems (such as subversion or git). As an example, consider a system for version management of files where a user can perform the following operations on documents: save them, access and modify them, undo one (or more) previous changes, restoring the previously saved version.

The following alphabet represents the user's actions: sv (for *save*), wr (for *write*, i.e. the document is opened and modified), ud (for a single *undo* operation), rb (for a *rollback* operation, where all the changes occurred since the previously saved version are discarded).

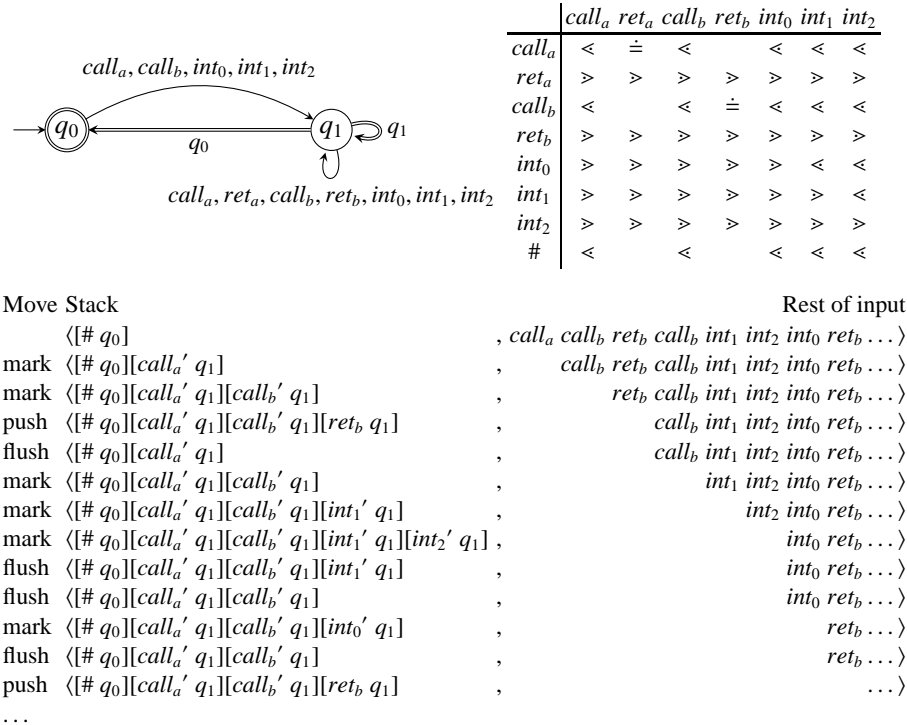
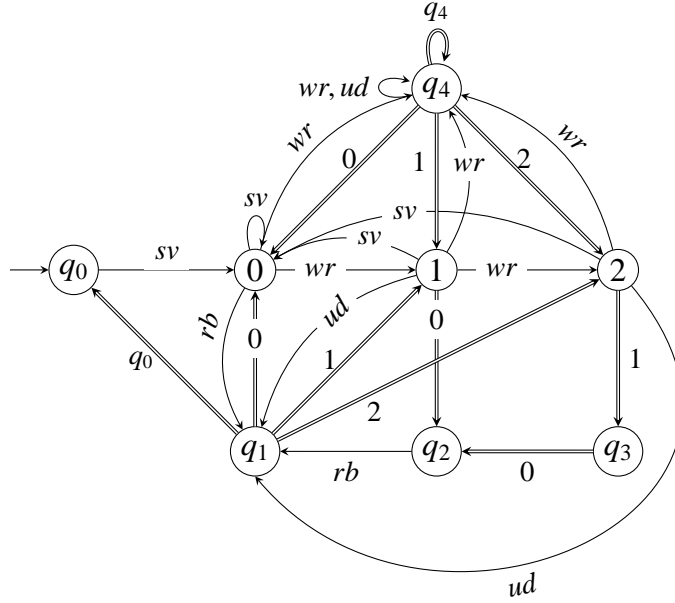


Fig. 2: Automaton, precedence matrix and example of computation for language of Example 2.

An ω OPBA which models the traces of possible actions of the user on a given document is a single-state automaton $\langle \Sigma, M, \{q\}, \{q\}, \delta \rangle$, where $\Sigma = \{sv, rb, wr, ud\}$, $\delta_{push}(q, a) = q$, $\forall a \in \Sigma$ and $\delta_{flush}(q, q) = q$ and its OPM is:

$$M = \begin{array}{c|cccc} & sv & rb & wr & ud \\ \hline sv & < & \doteq & < & \\ rb & > & > & > & > \\ wr & < & > & < & \doteq \\ ud & > & > & > & > \\ \# & < & & < & & \end{array}$$

Furthermore, one can even consider some specialized models of this system, that represent various patterns of user behavior. For instance, one in which the user regularly backs her work up, so that no more than N changes which are not undone (denoted wr as before) can occur between any two consecutive checkpoints sv (without any rollback rb between them). Figure 3 shows the corresponding ω OPBA with $N = 2$, with the same OPM M .

Fig. 3: ω OPBA of Example 3, with $N = 2$.

States 0, 1 and 2 denote respectively the presence of zero, one and two unmatched changes between two symbols sv . All states of the ω OPBA final.

An example of computation on the string $sv\ wr\ ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots$ is shown in Figure 4.

3.2 Operator precedence ω -languages and visibly pushdown ω -languages

Classical families of automata, like Visibly Pushdown Automata [1], imply several restrictions that hinder them from being able to deal with the concept of precedence among symbols. These restrictions make them unsuitable to define systems like those of Section 3.1, and in general all paradigms based on a model of priorities.

Noticeably, VPAs on infinite-length words are significantly extended by the class of OPAs, since VPAs introduce a rigid partitioning on the alphabet symbols which heavily constrains the possible relationships among them: any letter cannot assume a role dependent on the context (as an interrupt which can yield or take precedence over another one depending on the mutual priority), and this restriction has some consequences on their expressive power w.r.t ω OPLs. Actually, as it happens for finite-word languages [6,10], one can prove the following result.

Theorem 1. *The class of languages accepted by ω BVPA (nondeterministic Büchi visibly pushdown ω -automata) is a proper subset of that accepted by ω OPBA.*

Move Stack	Rest of input
$\langle [\# q_0] \rangle$, $sv\ wr\ ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0] \rangle$, $wr\ ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1] \rangle$, $ud\ rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
push $\langle [\# q_0][sv' 0][wr' 1][ud\ q_1] \rangle$, $rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
flush $\langle [\# q_0][sv' 0] \rangle$, $rb\ sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
push $\langle [\# q_0][sv' 0][rb\ q_1] \rangle$, $sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
flush $\langle [\# q_0] \rangle$, $sv\ wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0] \rangle$, $wr\ wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1] \rangle$, $wr\ ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1][wr' q_4] \rangle$, $ud\ sv\ wr\ rb\ wr\ sv\ \dots \rangle$
push $\langle [\# q_0][sv' 0][wr' 1][wr' q_4][ud\ q_4] \rangle$, $sv\ wr\ rb\ wr\ sv\ \dots \rangle$
flush $\langle [\# q_0][sv' 0][wr' 1] \rangle$, $sv\ wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1][sv' 0] \rangle$, $wr\ rb\ wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1][sv' 0][wr' 1] \rangle$, $rb\ wr\ sv\ \dots \rangle$
flush $\langle [\# q_0][sv' 0][wr' 1][sv' q_2] \rangle$, $rb\ wr\ sv\ \dots \rangle$
push $\langle [\# q_0][sv' 0][wr' 1][sv' q_2][rb\ q_1] \rangle$, $wr\ sv\ \dots \rangle$
flush $\langle [\# q_0][sv' 0][wr' 1] \rangle$, $wr\ sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1][wr' 2] \rangle$, $sv\ \dots \rangle$
mark $\langle [\# q_0][sv' 0][wr' 1][wr' 2][sv' 0] \rangle$, $\dots \rangle$
...	

Fig. 4: Example of computation for the specialized system of Example 3

The behavior of version management systems like those in Example 3 too cannot be modeled by ω VPAs since the shape of their matrix allows only one-to-one relationships between matching symbols (as do-undo actions on a single change, denoted wr and ud), whereas the return to a previous version, undoing all the possible sequence of changes performed in the meanwhile, is represented by a many-to-one relationship (holding among symbols wr and a single rb).

3.3 Other automata models for operator precedence ω -languages

There are several possibilities to define other classes of ω -languages. In order to do that we introduce the following general definition.

Definition 6. A nondeterministic operator precedence ω -automaton (ω OPA) is given by a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$, where Σ, Q, I, δ are defined as for OPAs; the operator precedence matrix M is restricted to be a $|\Sigma \cup \{\#\}| \times |\Sigma|$ array, since ω -words are not terminated by the delimiter $\#$; \mathcal{F} is an acceptance component, distinctive of the class (Büchi, Muller, ...) the automaton belongs to. Deterministic ω OPA are specified as for operator precedence automata on finite-length words.

A run is *successful* if it satisfies an acceptance condition on \mathcal{F} based on a specific recognizing mode. \mathcal{A} *accepts* $x \in \Sigma^\omega$ iff there is a successful run of \mathcal{A} on x . Furthermore, let the ω -language *recognized* by \mathcal{A} be $L(\mathcal{A}) = \{x \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } x\}$.

When \mathcal{F} is a subset $F \subseteq Q$, Definition 6 leads to Definition 5 of Büchi-operator precedence ω -automaton; ω OPBEA is a variant of ω OPBA obtained when using the following acceptance condition: a word is recognized if the automaton traverses final states with an empty stack infinitely often. Formally, a run \mathcal{S} of an ω OPBEA is successful iff there exists a state $q_f \in F$ such that configurations with stack $[\# q_f]$ occur infinitely often in \mathcal{S} .

Proposition 1. $\mathcal{L}(\omega$ OPBEA) $\subset \mathcal{L}(\omega$ OPBA).

Proof. The inclusion is trivial by definition. To see why it is proper, one can consider for instance the language L_{repbdd} (studied in [1]) consisting of infinite words on the alphabet $\{a, \underline{a}\}$, which can be interpreted as a language of calls and returns of a procedure a , with the further constraint that there is always a finite number of pending calls. A nondeterministic ω OPBA with final state acceptance condition can nondeterministically guess which is the prefix of the word containing the last pending call, and then recognizes the language $(L_{\text{Dyck}}(a, \underline{a}))^\omega$ of correctly nested words. An ω OPBEA cannot recognize this language. In fact, it may accept a word iff it reaches infinitely often a final configuration with empty stack during the parsing. However, the automaton is never able to remove all the input symbols piled on the stack since it cannot flush the pending calls interspersed among the correctly nested letters a , otherwise it would either introduce conflicts in the OPM or it would not be able to verify that they are in finite number.

The classical notion of acceptance for Muller automata can be likewise defined for ω OPAs.

Definition 7. A nondeterministic Muller-operator precedence automaton (ω OPMA) is an ω OPA $\langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$ whose acceptance component is a collection of subsets of Q , $\mathcal{F} = \mathcal{T} \subseteq 2^Q$, called the table of the automaton.

A run \mathcal{S} of an ω OPMA on an infinite word $x \in \Sigma^\omega$ is successful iff $\text{In}(\mathcal{S}) \in \mathcal{T}$, i.e. the set of states occurring infinitely often on the stack is a set in the table \mathcal{T} .

In the case of classical finite-state automata on infinite words, nondeterministic Büchi automata and nondeterministic Muller automata are equivalent and define the class of ω -regular languages. Traditionally, Muller automata have been introduced to provide an adequate acceptance mode for deterministic automata on ω -words. In fact, deterministic Büchi automata cannot recognize all ω -regular languages, whereas deterministic Muller automata are equivalent to nondeterministic Büchi ones [16].

For VPAs on infinite words, instead, the paper [1] showed that the classical determinization algorithm of Büchi automata into deterministic Muller automata is no longer valid, and deterministic Muller ω VPAs are strictly less powerful than nondeterministic Büchi ω VPAs. A similar relationship holds for ω OPAs too.

The relationships among languages recognized by the different classes of operator precedence ω -automata and visibly pushdown ω -languages are summarized in the structure of Figure 5, where ω DOPBA and ω DOPMA denote the classes of deterministic ω OPBAs and deterministic ω OPMAs respectively. The detailed proofs of the strict containment relations holding among the classes in Figure 5 are presented in [13, Chapter 4] and we do not report them here again for space reasons. In the following sections

we provide the proofs regarding the relationships between those classes which are not comparable (i.e., those linked with dashed lines in the figure), which are not included in [13].

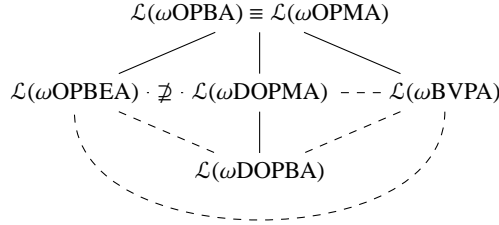


Fig. 5: Containment relations for ω OPLs. Solid lines denote strict inclusions; dashed lines link classes which are not comparable.

3.4 Comparison between $\mathcal{L}(\omega$ BVPA) and $\mathcal{L}(\omega$ OPBEA)

$\mathcal{L}(\omega$ BVPA) and $\mathcal{L}(\omega$ OPBEA) are not comparable.

- $\mathcal{L}(\omega$ BVPA) $\not\subseteq \mathcal{L}(\omega$ OPBEA)

Consider the language L_{repbdd} (studied in [1]) consisting of infinite words on the alphabet $\{a, \underline{a}\}$, which can be interpreted as a language of calls and returns of a procedure a , with the further constraint that there is only a finite number of pending calls. An ω BVPA can accept this language: it nondeterministically guesses which is the prefix of the string containing the last pending call, and it can subsequently recognize the language $(L_{\text{Dyck}}(a, \underline{a}))^\omega$ of correctly nested words.

An ω OPBEA automaton cannot recognize this language, as seen in the proof of Proposition 1.

- $\mathcal{L}(\omega$ BVPA) $\not\supseteq \mathcal{L}(\omega$ OPBEA)

Consider the system introduced in Example 4 of [10] which describes the stack management of a programming language able to handle nested exceptions. No ω BVPA can express the language of the infinite computations of this system because of the shape of the precedence matrix, which is not compatible with the matrix of a VPA.

The automaton presented in the figure of this Example 4, which is able to recognize this language, instead, can be interpreted as an ω OPBEA.

Note also that the same automaton can be considered as an ω OPBA: it is deterministic by construction, so there exists also an ω DOPBA able to model this system, and $\mathcal{L}(\omega$ BVPA) $\not\supseteq \mathcal{L}(\omega$ DOPBA). Moreover, since $\mathcal{L}(\omega$ DOPBA) $\subseteq \mathcal{L}(\omega$ DOPMA), an automaton ω DOPMA can recognize it too; thus $\mathcal{L}(\omega$ BVPA) $\not\supseteq \mathcal{L}(\omega$ DOPMA).

3.5 Comparison between $\mathcal{L}(\omega\text{BVPA})$ and $\mathcal{L}(\omega\text{DOPBA})$

$\mathcal{L}(\omega\text{BVPA})$ and $\mathcal{L}(\omega\text{DOPBA})$ are not comparable.

- $\mathcal{L}(\omega\text{BVPA}) \not\subseteq \mathcal{L}(\omega\text{DOPBA})$

Consider the language on the alphabet $\Sigma = \{a, b\}$:

$$L_1 = \{\alpha \in \Sigma^\omega : \alpha \text{ contains finitely many letters } a\} \quad (3)$$

It can be recognized by an ωBVPA , but no ωDOPBA can accept it.

In fact, an ωBVPA can recognize words of L_1 finding nondeterministically the last letter a in a word and then reading suffix b^ω .

The proof that no ωDOPBA can recognize L_1 resembles the classical proof (see e.g. [16]) that deterministic Büchi finite-state automata are strictly weaker than nondeterministic Büchi finite-state ones. We outline here the proof for the sake of completeness.

Assume that there exists an ωDOPBA \mathcal{B} which recognizes L_1 .

Notice that, in general, according to the definition of push/mark/flush moves of an operator precedence automaton (finite or ω), given any configuration $C = \langle \beta, w \rangle$, the state piled up at the top of the stack with a transition $\langle \beta, w \rangle \vdash \langle \beta', w' \rangle$, namely $\text{state}(\beta')$, is exactly the state reached by the automaton on its state-graph. Thus, during a run on a word $x \in \Sigma^\omega$, configurations with stack β_i with $\text{state}(\beta_i) \in F$ occur infinitely often iff the automaton visits infinitely often states in F in its graph. Now, the infinite word $x = b^\omega$ belongs to L_1 , since it contains no (and then a finite number of) letters a . Then, there exists a unique run of \mathcal{B} on this string which visits infinitely often final states. Let b^{n_1} be the prefix read by \mathcal{B} until the first visited final state.

But also $b^{n_1}ab^\omega$ belongs to L_1 , hence there exists a final state reached reading the prefix $b^{n_1}ab^{n_2}$, for some $n_2 \in \mathbb{N}$.

In general, one can find a sequence of finite words $b^{n_1}ab^{n_2} \dots ab^{n_k}$, ($k \geq 1$) such that the automaton has a unique run on them, and for each such runs it reaches a final state (placing it at the top of the stack) after reading every prefix $b^{n_1}ab^{n_2} \dots ab^{n_i}$, $\forall i \leq k$. Therefore, there exists a (unique) run of \mathcal{A} on the ω -word $w = b^{n_1}ab^{n_2} \dots$ such that \mathcal{A} visits infinitely often final states, and thus reaches infinitely often configurations $C = \langle \beta, w \rangle$ with $\text{state}(\beta) \in F$.

However, w cannot be accepted by \mathcal{B} since it contains infinitely many letters a , and this is a contradiction.

- $\mathcal{L}(\omega\text{BVPA}) \not\supseteq \mathcal{L}(\omega\text{DOPBA})$

See Section 3.4

3.6 Comparison between $\mathcal{L}(\omega\text{BVPA})$ and $\mathcal{L}(\omega\text{DOPMA})$

$\mathcal{L}(\omega\text{BVPA})$ and $\mathcal{L}(\omega\text{DOPMA})$ are not comparable.

- $\mathcal{L}(\omega\text{BVPA}) \not\subseteq \mathcal{L}(\omega\text{DOPMA})$

No ωDOPMA can recognize the language L_{repbdd} (the proof can be found in [13]), whereas an ωBVPA can accept it (see [1]).

- $\mathcal{L}(\omega\text{BVPA}) \not\supseteq \mathcal{L}(\omega\text{DOPMA})$

See Section 3.4

3.7 Comparison between $\mathcal{L}(\omega\text{OPBEA})$ and $\mathcal{L}(\omega\text{DOPBA})$

$\mathcal{L}(\omega\text{OPBEA})$ and $\mathcal{L}(\omega\text{DOPBA})$ are not comparable.

- $\mathcal{L}(\omega\text{OPBEA}) \not\subseteq \mathcal{L}(\omega\text{DOPBA})$

Language L_1 (Equation 3) cannot be recognized by an ωDOPBA (see Section 3.5), but there exists an ωOPBEA accepting it, depicted in Figure 6 along with its precedence matrix (where $\circ \in \{<, \doteq, >\}$ can be any precedence relation):

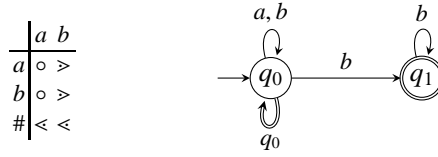


Fig. 6: ωOPBEA recognizing $L_1 = \{\alpha \in \Sigma^\omega : \alpha \text{ contains finitely many letters } a\}$ and its OPM.

- $\mathcal{L}(\omega\text{OPBEA}) \not\supseteq \mathcal{L}(\omega\text{DOPBA})$

Let L_2 be the language $a^2 L_3^\omega$ with $L_3 = \{a^k b^k \mid k \geq 1\}$ and where, in general, for a set of finite words $L \subseteq A^*$, one defines $L^\omega = \{\alpha \in A^\omega \mid \alpha = w_0 w_1 \dots \text{ with } w_i \in L \text{ for } i \geq 0\}$.

No ωOPBEA can recognize this language. Indeed, words in L_3 can be recognized only with the OPM M depicted in Figure 7, where $\circ \in \{<, \doteq, >\}$ can be any precedence relation: clearly, using any other OPM there exist words in L_3 and $L_2 = a^2 L_3^\omega$ which could not be recognized. Thus, because of the OP relation $a < a$, an ωOPBEA piles up on the stack the first sequence a^2 of a word and cannot remove it afterwards; hence it cannot empty the stack infinitely often to accept a string in L_2 .

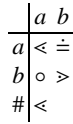
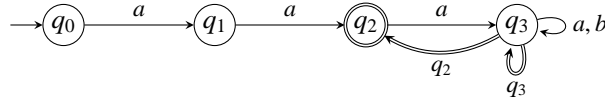


Fig. 7: OPM for language L_2 of Section 3.7.

There is, however, an ωDOPBA that recognizes such a language (Figure 8). Incidentally notice that, since $\mathcal{L}(\omega\text{DOPBA}) \subseteq \mathcal{L}(\omega\text{DOPMA})$, an automaton ωDOPMA can recognize it too; thus $\mathcal{L}(\omega\text{OPBEA}) \not\supseteq \mathcal{L}(\omega\text{DOPMA})$.

Fig. 8: ω DOPBA recognizing language L_2 of Section 3.7.

4 Closure properties and emptiness problem

$\mathcal{L}(\omega\text{OPBA})$ enjoys all closure and decidability properties necessary to perform model checking; thus thanks to their greater expressive power, we believe that they represent a truly promising formalism for infinite-state model-checking.

In the first part of this section we focus on the most interesting closure properties of ω OPAs, which are summarized in Table 1, where they are compared with the properties enjoyed by VPAs on infinite-length words. Binary operations are considered between languages with compatible OPMs.

	$\mathcal{L}(\omega\text{DOPBA})$	$\mathcal{L}(\omega\text{DOPMA})$	$\mathcal{L}(\omega\text{OPBA}) \equiv \mathcal{L}(\omega\text{OPMA})$	$\mathcal{L}(\omega\text{BVPA})$ [1]
Intersection	Yes	Yes	Yes	Yes
Union	Yes	Yes	Yes	Yes
Complement	No	Yes	Yes	Yes
$L_1 \cdot L_2$	No	No	Yes	Yes

Table 1: Closure properties of families of ω -languages. ($L_1 \cdot L_2$ denotes the concatenation of a language of finite-length words L_1 and an ω -language L_2).

Closure properties for ω DOPBAs (under complement and concatenation with an OPL) and ω DOPMAs are not discussed here because of space reasons, but they resemble proofs for classical families of ω -automata and can anyhow be found in [13]. Closure properties for ω DOPBAs under intersection and union are presented in Section 4.1.

We consider in detail the main family ω OPBA. This class is closed under Boolean operations between languages with compatible precedence matrices and under concatenation with a language of finite words accepted by an OPA. The emptiness problem is decidable for ω OPAs in polynomial time because they can be interpreted as pushdown automata on infinite-length words: e.g. [5] shows an algorithm that decides the alternation-free modal μ -calculus for context-free processes, with linear complexity in the size of the system's representation; thus the emptiness problem for the intersection of the language recognized by a pushdown process and the language of a given property in this logic is decidable. Closures under intersection and union hold for ω OPBAs as for classical ω -regular languages and can be proved in a similar way [13]. Closures under complementation and concatenation required novel investigation techniques.

Closure under concatenation

For classical families of automata (on finite or infinite-length words) the closure of the class of languages they recognize with respect to the operation of concatenation is tra-

ditionally proved resorting to a Thompson-like construction: given two automata that recognize languages of a given class, an automaton which accepts the concatenation of these languages is generally defined so that it may simulate the moves of the first automaton while reading the first word of the concatenation and, once it reaches some final state, it switches to the initial states of the second automaton to begin the recognition of words of the second language.

This construction, however, is not adequate for the concatenation of a language of finite words recognized by a classical OPA and an ω OPL (recognized by an ω OPBA). In fact, a classical OPA accepts a finite word by reaching a final state and by emptying its stack thanks to the ending delimiter $\#$. As regards the concatenation of a language recognized by an OPA and an ω -language (accepted by an ω OPBA) whose words are not ended by $\#$, this condition is not necessarily guaranteed and it might be not possible to complete the recognition of a word of the first language simulating the behavior of its OPA according to the acceptance condition by final state and empty stack. As an example, for a language $L_1 \subseteq \Sigma^*$ and an ω -language $L_2 = \{a^\omega\}$ with compatible precedence matrices such that all letters of the alphabet yield precedence to symbol a (i.e. $b < a, \forall b \in \Sigma$), the symbols still on the stack after reading words in L_1 cannot be removed with flush moves before or during the parsing of the second word in the concatenation, since the precedence relation $<$ implies that the letters read are only pushed on the stack. Thus, the stack cannot be emptied after the reading of the first word, and this prevents to check if it actually belongs to the first language of the concatenation.

After reading the first finite word in the concatenation, it is not even possible to determine whether this word is accepted by checking if in its OPA there exists an ongoing run on it that could lead to a final state by flush moves induced by a potential delimiter $\#$, since this control would require to know the states already reached and piled on the stack, which are not visible without emptying the stack itself.

Closure under concatenation for the class of languages accepted by ω OPBAs with a language of finite words accepted by an OPA could be proved similarly as for classical automata if it were possible to recognize finite words by an OPA without emptying the stack and without even performing any flush move induced by symbol $\#$ immediately after reading the word; in this way the acceptance could be completed even when the words of the second language prevent emptying the stack.

To this aim, a possible solution is to introduce a variant of the semantics of the transition relation and of the acceptance condition for OPAs on finite-length words: a string is accepted if the automaton reaches a final state right at the end of the parsing of the whole word, and does not perform any flush move determined by the ending delimiter $\#$ to empty the stack; thus it stops just after having put the last symbol of x on the stack. Precisely, the semantics of the transition relation differs from the definition of classical OPAs in that, once a configuration with the endmarker as lookahead is reached, the computation cannot evolve in any subsequent configuration, i.e. a flush move $C \vdash C_1$ with $C = \langle B_1 B_2 \dots B_n, x\# \rangle$ and $\text{symbol}(B_n) > y\#$ is performed only if $y \neq \varepsilon$. The language accepted by this variant of the automaton (denoted as \tilde{L}) is the set of words:

$$\tilde{L}(\mathcal{A}) = \{x \mid \langle \# q_I \rangle, x\# \rangle^* \langle \gamma[a q_F], \# \rangle, q_I \in I, q_F \in F, \gamma \in \Gamma^*, a \in \Sigma \cup \{\#\}\}$$

We emphasize that, unlike normal acceptance by final state of a pushdown automaton, which can perform a number of ε -moves after reaching the end of a string and accept if just one of the visited states is final, this type of automaton cannot perform any (flush) move after reaching the endmarker through the last look-ahead.

Nevertheless, the variant and the classical definition of OPA are equivalent, as the following statements (Lemma 1 and Statement 1) prove.

Lemma 1. *Let \mathcal{A}_1 be a nondeterministic OPA defined on an OP alphabet (Σ, M) with s states. Then there exists a nondeterministic OPA \mathcal{A}_2 with the same precedence matrix as \mathcal{A}_1 and $O(|\Sigma|s^2)$ states such that $L(\mathcal{A}_1) = \tilde{L}(\mathcal{A}_2)$.*

To build such a variant \mathcal{A}_2 we need some further notation. Consider a word of finite length w which is preceded by a delimiter $\#$ but which is not ended with such a symbol. Define a chain in a word w as *maximal* if it does not belong to a larger composed chain. In a word of finite length preceded and ended by $\#$ only the outer chain $\langle^\# w^\# \rangle$ is maximal.

An *open chain* is a sequence of symbols $b_0 < a_1 \doteq a_2 \doteq \dots \doteq a_n$, for $n \geq 1$.

The *body* of a chain $\langle^a x^b \rangle$, simple or composed, is the word x . A letter $a \in \Sigma$ in a word $\#w^\#$ with $w \in \Sigma^*$ or $\#w$ with $w \in \Sigma^\omega$ is *pending* if it does not belong to the body of a chain, i.e., once pushed on the stack when it is read, it will never be flushed afterwards.

A word w which is preceded but not ended by a delimiter $\#$ can be factored in a unique way as a sequence of bodies of maximal chains w_i and pending letters a_i as $\# w = \# w_1 a_1 w_2 a_2 \dots w_n a_n$ where $\langle^{a_{i-1}} w_i^{a_i} \rangle$ are maximal chains and each w_i can be possibly missing, with $a_0 = \#$ and $\forall i : 1 \leq i \leq n-1 \ a_i < a_{i+1}$ or $a_i \doteq a_{i+1}$.

In general, during the parsing of word w , the symbols of the string are put on the stack and, whenever a chain is recognized, the letters of its body are flushed away.

Hence, after the parsing of the whole word the stack contains only the symbols $\# a_1 a_2 \dots a_n$ and is structured as a sequence of open chains. Let k be the number of open chains and denote by $a_1 = a_{i_1}, a_{i_2}, \dots, a_{i_k}$ their starting symbols, then the stack contains:

$$\# < a_{i_1} = a_1 \doteq a_2 \doteq \dots < a_{i_2} \doteq a_{i_2+1} \dots < a_{i_3} \doteq a_{i_3+1} \dots < a_{i_k} \doteq a_{i_k+1} \dots \doteq a_n$$

When a word w is parsed by a classical OPA, the automaton performs a series of flush moves at the end of the string due to the presence of the final symbol $\#$. These moves progressively empty the stack, removing one by one the open chains and, for each such flush, they update the state of the automaton on the basis of the symbols which delimit the portion of the stack to be removed, which correspond to the state symbols at the end of the current open chain and at the end of the preceding open chain. The run is accepting if it leads to a final state after the flush moves.

As an example, the transition sequence below shows the flush moves of a classical OPA when it reaches the position of a_n :

$$\begin{aligned} & \langle [\# q_1][a_{i_1}' q_2][a_2 q_3] \dots [a_{i_2-1} q_{i_2}][a_{i_2}' q_{i_2+1}] \dots [a_{i_3-1} q_{i_3}] \dots [a_{i_k-1} q_{i_k}][a_{i_k}' q_{i_k+1}] \dots [a_n q_{n+1}], \# \rangle \\ & \vdash \langle [\# q_1][a_{i_1}' q_2][a_2 q_3] \dots [a_{i_2-1} q_{i_2}][a_{i_2}' q_{i_2+1}] \dots [a_{i_3-1} q_{i_3}] \dots [a_{i_k-1} \hat{q}_{i_k} = \delta_{\text{flush}}(q_{n+1}, q_{i_k})], \# \rangle \\ & \stackrel{*}{\vdash} \langle [\# q_1][a_{i_1}' q_2][a_2 q_3] \dots [a_{i_2-1} q_{i_2}][a_{i_2}' q_{i_2+1}] \dots [a_{i_3-1} \hat{q}_{i_3} = \delta_{\text{flush}}(\hat{q}_{i_4}, q_{i_3})], \# \rangle \end{aligned}$$

$$\begin{aligned} &\vdash \langle [\# q_1][a_{i_1}' q_2][a_2 q_3] \dots [a_{i_2-1} \hat{q}_{i_2} = \delta_{\text{flush}}(\hat{q}_{i_3}, q_{i_2})], \# \rangle \\ &\vdash \langle [\# \hat{q}_1 = \delta_{\text{flush}}(\hat{q}_2, q_1)], \# \rangle \end{aligned}$$

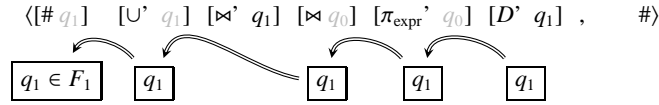
A nondeterministic automaton that, unlike classical OPAs, does not resort to the delimiter $\#$ for the recognition of a string may guess nondeterministically the ending point of each open chain on the stack and may guess how, in an accepting run, the states in these points of the stack would be updated if the final flush moves were progressively performed. The automaton must behave as if, at the same time, it simulates two snapshots of the accepting run of a classical OPA: a move during the parsing of the string and a step during the final flush transitions which will later on empty the stack, leading to a final state. To this aim, the states of a classical OPA are augmented with an additional component to store the necessary information.

In the initial configuration, the symbol at the bottom of the stack comprises, along with an initial state q of the original OPA \mathcal{A}_1 , an additional state, say q_F , which represents a final state of \mathcal{A}_1 . The additional component is propagated until the automaton nondeterministically identifies the first pending letter, which represents the beginning of the first open chain; at this time the component is updated with a new state chosen so that there exists a move from it in \mathcal{A}_1 that can flush and replace the state at the bottom of the stack with the final one q_F (notice that if the beginning letter of the word is not a pending letter – i.e., the prefix of the word is a maximal chain – after completing the parsing of the chain, the initial state q will be flushed and replaced on the bottom of the stack by a new state, say r , like in a classical OPA; in this case the last component added after reading the pending letter is chosen so that there exists a move in the graph of \mathcal{A}_1 that can flush and replace the state r with q_F). Then, similarly, the additional component is propagated until the ending point of each open chain, until the conclusion of the parsing; while reading the pending letter that represents the beginning of the successive open chain the automaton augments the new state on the stack with a placeholder chosen so that there is a flush move in \mathcal{A}_1 from it that can replace the state at the end of the previous open chain with the additional component previously stacked, thus allowing a backward path of flush moves from each ending point of an open chain to the previous one, up to the final state initially stacked. If the forward path consisting of moves during the parsing of the string and this backward path of flush moves can consistently meet and be rejoined when the parsing of the input string stops, then they constitute an entire accepting run of the classical OPA.

A variant OPA \mathcal{A}_2 equivalent to a given OPA \mathcal{A}_1 thus may be defined so that, after reading each prefix of a word, it reaches a final state whenever, if the word were completed in that point with $\#$, \mathcal{A}_1 could reach an accepting state with a sequence of flush moves. In this way, \mathcal{A}_2 can guess in advance which words may eventually lead to an accepting state of \mathcal{A}_1 , without having to wait until reading the delimiter $\#$ and to perform final flush moves.

Example 4. Consider the computation of the OPA in Example 1. If we consider the input word of this computation without the ending marker $\#$, then the sequence of pending letters on the stack, after the automaton puts on the stack the last symbol D , is $\# < \cup < \bowtie \doteq \bowtie < \pi_{\text{expr}} < D$. There are four open chains with starting symbols \cup , \bowtie , π_{expr} , D , hence the computation ends with four consecutive flush moves

determined by the delimiter $\#$. The following figure shows the configuration just before looking ahead at the symbol $\#$. The states (depicted within a box) at the end of the open chains are those placeholders that an equivalent variant OPA should guess in order to find in advance the last flush moves $q_1 = \boxed{q_1} \xRightarrow{q_0} \boxed{q_1} \xRightarrow{q_0} \boxed{q_1} \xRightarrow{q_1} \boxed{q_1} \xRightarrow{q_1} \boxed{q_1 \in F_1}$ of the accepting run.



The corresponding configuration of the variant OPA, with the augmented states, would be:

$$\langle [\# q_1, \boxed{q_1}] \quad [\cup' q_1, \boxed{q_1}] \quad [\bowtie' q_1, \boxed{q_1}] \quad [\bowtie q_0, \boxed{q_1}] \quad [\pi_{\text{expr}}' q_0, \boxed{q_1}] \quad [D' q_1, \boxed{q_1}] \quad , \quad \# \rangle$$

We are now ready to formally prove Lemma 1.

Proof. Let $\mathcal{A}_1 = \langle \Sigma, M, Q_1, I_1, F_1, \delta_1 \rangle$ and define $\mathcal{A}_2 = \langle \Sigma, M, Q_2, I_2, F_2, \delta_2 \rangle$ as follows.

- $Q_2 = \{B, Z, U\} \times \hat{\Sigma} \times Q_1 \times Q_1$, where $\hat{\Sigma} = \Sigma \cup \{\#\}$.

Hence, a state $\langle x, a, q, p \rangle$ of \mathcal{A}_2 is a tuple whose first component denotes a non-deterministic guess for the next symbol of the word to be read, i.e., a pending letter starting an open chain (Z), or a pending letter within an open chain (U), or a symbol within a maximal chain (B). The second and third components of a state represent, respectively, the lookback letter a read to reach the state, and the current state q in \mathcal{A}_1 . To see the meaning of the last component, consider an accepting run of \mathcal{A}_1 and let q be the current state just before a mark move is going to be performed at the beginning of an open chain; also let r be the state reached by the mark move and s be the state on top of the stack when this open chain is to be flushed replacing q with a new state p . Then, in the same position of the corresponding run of \mathcal{A}_2 , the current state would be $\langle Z, a, q, p \rangle \in Q_2$ and state $\langle x, a, r, s \rangle \in Q_2$ will be reached by \mathcal{A}_2 , i.e., the last component p represents a guess about the state that will replace q in \mathcal{A}_1 when the starting open chain will be flushed. Hence we can consider only states $\langle Z, a, q, p \rangle \in Q_2$ such that $r \xRightarrow{q} p$ in \mathcal{A}_1 for some $r \in Q_1$. In all other positions the last component of the states in Q_2 is simply propagated.

- $I_2 = \{\langle x, \#, q, q_F \rangle \mid x \in \{Z, B\}, q \in I_1, q_F \in F_1\}$
- $F_2 = \{\langle Z, a, q, q \rangle \mid q \in Q_1, a \in \hat{\Sigma}\}$

- The transition function is defined as the union of two disjoint functions.

The push transition function $\delta_{2\text{push}} : Q_2 \times \Sigma \rightarrow 2^{Q_2}$ is defined as follows, where $p, q, r, s \in Q_1$, $a \in \hat{\Sigma}$, and $b, c \in \Sigma$.

- *Mark of a pending letter at the beginning of an open chain.* If $a \prec b$ then:

$$\delta_{2\text{push}}(\langle Z, a, q, p \rangle, b) = \left\{ \langle x, b, r, s \rangle \mid x \in \{B, Z, U\}, q \xrightarrow{b} r, s \xRightarrow{q} p \text{ in } \mathcal{A}_1 \right\}$$

- *Push of a pending letter within an open chain.* If $a \doteq b$ then:

$$\delta_{2\text{push}}(\langle U, a, q, p \rangle, b) = \left\{ \langle x, b, r, p \rangle \mid x \in \{B, Z, U\}, q \xrightarrow{b} r \text{ in } \mathcal{A}_1 \right\}$$

- *Push/mark of a symbol of a maximal chain.*

$$\delta_{2\text{push}}(\langle B, a, q, p \rangle, b) = \left\{ \langle B, b, r, p \rangle \mid q \xrightarrow{b} r \text{ in } \mathcal{A}_1 \right\}$$

Notice that the second and third components of the states computed by $\delta_{2\text{push}}$ are independent of the first component of the starting state.

The flush transition function $\delta_{2\text{flush}} : Q_2 \times Q_2 \rightarrow 2^{Q_2}$ can be executed only within a maximal chain since there are no flush determined by the ending delimiter:

$$\delta_{2\text{flush}}(\langle B, b, q, s \rangle, \langle B, c, p, s \rangle) = \left\{ \langle x, c, r, s \rangle \mid x \in \{B, Z, U\}, q \xRightarrow{p} r \text{ in } \mathcal{A}_1 \right\}$$

All other moves lead to an error state.

The automata \mathcal{A}_1 and \mathcal{A}_2 recognize the same language, $L(\mathcal{A}_1) = \widetilde{L}(\mathcal{A}_2)$.

Let us prove first $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. Let $w \in L(\mathcal{A}_1)$ be a finite-length word. Then there exist a support $q \xrightarrow{w} q'$ in \mathcal{A}_1 with $q \in I_1$ and $q' \in F_1$. If $w = w_1 a_1 w_2 a_2 \dots w_n a_n \in L(\mathcal{A}_1)$ where a_i are pending letters and w_i are maximal chains, let k be the number of open chains that remain on the stack after the parsing of the last symbol in Σ of w , and let $a_{i_1} = a_1, a_{i_2}, \dots, a_{i_k}$ be their starting symbols. Also, for every $i = 2, \dots, n$, let $t(i)$ be the greatest index t such that $i_t < i$, i.e., a_i is within the $t(i)$ -th open chain starting with $a_{i_{t(i)}}$. In particular, for $i = n$, if $a_{n-1} \leq a_n$ then $i_k = n$, otherwise $t(n) = k$.

Then the above support for w can be decomposed as

$$q = \widetilde{q}_0 \xrightarrow{w_1} q_1 \xrightarrow{a_1} \widetilde{q}_1 \xrightarrow{w_2} q_2 \xrightarrow{a_2} \dots \xrightarrow{w_n} q_n \xrightarrow{a_n} \widetilde{q}_n = p_k \quad (4)$$

$$\widetilde{q}_n = p_k \xRightarrow{q_{i_k}} p_{k-1} \xRightarrow{q_{i_{k-1}}} p_{k-2} \implies \dots \implies p_2 \xRightarrow{q_{i_2}} p_1 \xRightarrow{q_{i_1}=q_1} p_0 = q'$$

where $q_i = \widetilde{q}_{i-1}$ if $w_i = \varepsilon$ for $i = 1, 2, \dots, n$. Notice that, for every t , q_{i_t} is the state reached in this path before the mark move that pushes symbol a_{i_t} on the stack; moreover, when the open chain starting with a_{i_t} is to be flushed, the current state is p_t and then state q_{i_t} is replaced with p_{t-1} on top of the stack.

Starting with state $\langle Z, \#, q_1, p_0 \rangle$ if $w_1 = \varepsilon$ or with $\langle B, \#, \widetilde{q}_0, p_0 \rangle \xrightarrow{w_1} \langle Z, \#, q_1, p_0 \rangle$ if $w_1 \neq \varepsilon$, an accepting computation of \mathcal{A}_2 can be built on the basis of the following facts:

- Since $q_1 \xrightarrow{a_1} \widetilde{q}_1$ and $p_1 \xRightarrow{q_1} p_0$ in \mathcal{A}_1 , then $\delta_{2\text{push}}(\langle Z, \#, q_1, p_0 \rangle, a_1) \ni \langle x, a_1, \widetilde{q}_1, p_1 \rangle$ in \mathcal{A}_2 for $x \in \{U, Z\}$. This is a mark move that can be applied at the beginning of the first open chain starting with a_1 , where p_1 is the guess about the state that will be reached before such open chain will be flushed.
- In general, for every t , since $q_{i_t} \xrightarrow{a_{i_t}} \widetilde{q}_{i_t}$ and $p_t \xRightarrow{q_{i_t}} p_{t-1}$ in \mathcal{A}_1 , then $\delta_{2\text{push}}(\langle Z, a_{i_{t-1}}, q_{i_t}, p_{t-1} \rangle, a_{i_t}) \ni \langle x, a_{i_t}, \widetilde{q}_{i_t}, p_t \rangle$ for $x \in \{U, Z\}$. This is a mark move that can be applied at the beginning of the t -th open chain starting with a_{i_t} , where p_t is the guess about the state that will be reached before such open chain will be flushed. In particular, if $i_k = n$, we can reach state $\langle Z, a_n, \widetilde{q}_n, p_k \rangle$ which is final in \mathcal{A}_2 since $q_n = p_k$.

- For every maximal chain w_i of w (with $i \geq 2$) consider its support $\xrightarrow{a_{i-1}} \tilde{q}_{i-1} \xrightarrow{w_i} q_i$ in (4). Then in \mathcal{A}_2 we have the sequence of moves “summarized” (with a natural overloading of the notation) by $\delta_2(\langle B, a_{i-1}, \tilde{q}_{i-1}, p_{t(i)} \rangle, w_i) \ni \langle x, a_{i-1}, q_i, p_{t(i)} \rangle$, where $x \in \{U, Z\}$. Notice that the last component of the states does not change because we are within a maximal chain. In particular, during the parsing of w_i the last component is equal to $p_{t(i)}$, as guessed by the mark move at the beginning of the current open chain.
- For every $i \notin \{i_1, i_2, \dots, i_k\}$, since $\delta_{1\text{push}}(q_i, a_i) \ni \tilde{q}_i$, then $\delta_{2\text{push}}(\langle U, a_{i-1}, q_i, p_{t(i)} \rangle, a_i)$ contains $\langle x, a_i, \tilde{q}_i, p_{t(i)} \rangle$, for $x \in \{B, Z, U\}$. In particular, if $n \neq i_k$, then $t(n) = k$ and for $i = n$ we can reach state $\langle Z, a_n, \tilde{q}_n, p_k \rangle$ which is final in \mathcal{A}_2 since $q_n = p_k$.

Thus, by composing in the right order the previous moves, one can obtain an accepting computation for w in \mathcal{A}_2 .

Conversely, to prove that $\tilde{L}(\mathcal{A}_2) \subseteq L(\mathcal{A}_1)$, consider a finite word $w \in L(\mathcal{A}_2)$. Then there exists a successful run of \mathcal{A}_2 on w . Let w be factorized as above; then the accepting run for w can be decomposed as

$$\pi_0 \xrightarrow{w_1} \rho_1 \xrightarrow{a_1} \pi_1 \xrightarrow{w_2} \rho_2 \dots \rho_i \xrightarrow{a_i} \pi_i \xrightarrow{w_{i+1}} \dots \xrightarrow{w_n} \rho_n \xrightarrow{a_n} \pi_n$$

where $\pi_i, \rho_i \in Q_2$, $\rho_i = \pi_{i-1}$ if $w_i = \varepsilon$, $\pi_0 \in I_2$ and $\pi_n \in F_2$. By projecting this path on the third component of states π_i and ρ_i (given by, say, p_i and $r_i \in Q_1$), we obtain a path in \mathcal{A}_1 labelled by w . This path is not accepting because there are open chains left on the stack that need flushing, but we can complete this path arguing by induction on the structure of maximal chains according to the definition of δ_2 . More formally, one can verify that Q_1 contains suitable states p_i (for $0 \leq i \leq n$), r_i (for $1 \leq i \leq n$), s_i (for $1 \leq i \leq k$), with $r_i = p_{i-1}$ whenever $w_i = \varepsilon$, such that the following facts hold.

- $\pi_0 \in I_2$, hence $\pi_0 = \langle x_0, \#, p_0, s_0 \rangle$, with $p_0 \in I_1$ and $s_0 \in F_1$; x_0 is B if $w_1 \neq \varepsilon$, otherwise $x_0 = Z$.
- $\pi_0 \xrightarrow{w_1} \rho_1$ in \mathcal{A}_2 implies that the last component of state π_0 is propagated through chain w_1 without change; hence $\rho_1 = \langle Z, \#, r_1, s_0 \rangle$ with $p_0 \xrightarrow{w_1} r_1$ in \mathcal{A}_1 .
- $\rho_1 \xrightarrow{a_1} \pi_1$ is a mark move of \mathcal{A}_2 at the beginning of an open chain, and this implies that the last component of π_1 is new; hence we have $\pi_1 = \langle x_1, a_1, p_1, s_1 \rangle$ with $r_1 \xrightarrow{a_1} p_1$ and $s_1 \xrightarrow{r_1} s_0$ in \mathcal{A}_1 ; the first component is $x_1 = B$ if $w_2 \neq \varepsilon$ otherwise x_1 equals Z or U according to whether a_2 starts an open chains or not, respectively,
- The flush moves within $\pi_i \xrightarrow{w_{i+1}} \rho_{i+1}$ for $1 \leq i < i_2$, and the push moves within an open chain $\rho_i \xrightarrow{a_i} \pi_i$ for $1 < i < i_2$ propagate with no change the last component of states. Hence $\rho_i = \langle U, a_{i-1}, r_i, s_1 \rangle$ and $\pi_i = \langle x_i, a_i, p_i, s_1 \rangle$ with $p_{i-1} \xrightarrow{w_i} r_i \xrightarrow{a_i} p_i$ in \mathcal{A}_1 . The first component is $x_i = B$ if $w_i \neq \varepsilon$ otherwise $x_i = Z$ for $i = i_2 - 1$ and $x_i = U$ in the other cases.
- $\rho_{i_2} \xrightarrow{a_{i_2}} \pi_{i_2}$ is a mark move of \mathcal{A}_2 at the beginning of an open chain, and this implies that the last component of π_{i_2} is new; hence we have $\pi_{i_2} = \langle x_{i_2}, a_{i_2}, p_{i_2}, s_2 \rangle$ with $r_{i_2} \xrightarrow{a_{i_2}} p_{i_2}$ and $s_2 \xrightarrow{r_{i_2}} s_1$ in \mathcal{A}_1 . The first component is $x_{i_2} = B$ if $w_{i_2} \neq \varepsilon$ otherwise x_{i_2} equals Z or U according to whether $a_{i_2} + 1$ starts an open chains or not, respectively.

– Similarly for the following moves in the run.

In general, we get

$$\begin{aligned}
 \rho_i &= \langle y_i, a_{i-1}, r_i, s_{t(i)} \rangle && \text{for every } i = 1, 2, \dots, n, \\
 \pi_i &= \langle x_i, a_i, p_i, s_{t(i)} \rangle && \text{for every } i \notin \{i_1, i_2, \dots, i_k\}, \\
 \pi_{i_t} &= \langle x_{i_t}, a_{i_t}, p_{i_t}, s_{t_i} \rangle && \text{for every } t = 1, 2, \dots, k, \\
 \text{with } r_i &\xrightarrow{a_i} p_i, s_t \xrightarrow{r_{i_t}} s_{t-1}, p_{i-1} \xrightarrow{w_i} r_i \text{ in } \mathcal{A}_1 \\
 &\text{and } y_i \in \{Z, U\}, x_i \in \{B, Z, U\} && \text{for every } i \text{ and } t.
 \end{aligned}$$

By convention, $a_0 = \#$. For $i = n$ we have $n = i_k$ or $t(n) = k$, hence $\pi_n = \langle x_n, a_n, p_n, s_k \rangle$, and $p_n = s_k$ and $x_n = Z$ since $\pi_n \in F_2$. Thus, in \mathcal{A}_1 there is an accepting run

$$\begin{aligned}
 I_1 \ni p_0 &\xrightarrow{w_1} r_1 \xrightarrow{a_1} p_1 \xrightarrow{w_2} r_2 \dots r_i \xrightarrow{a_i} p_i \xrightarrow{w_{i+1}} \dots \xrightarrow{w_n} r_n \xrightarrow{a_n} p_n = s_k \\
 p_n = s_k &\xrightarrow{r_{i_k}} s_{k-1} \xrightarrow{r_{i_{k-1}}} s_{k-2} \implies \dots \implies s_2 \xrightarrow{r_{i_2}} s_1 \xrightarrow{r_{i_1}=r_1} s_0 \in F_1
 \end{aligned}$$

and this concludes the proof of the lemma. \square

The next Statement, although not necessary to prove closure under concatenation of $\mathcal{L}(\omega\text{OPBA})$, completes the proof of equivalence between traditional and variant OPAs, showing how to define, for any variant OPA, a classical OPA which recognizes the same language.

Statement 1 Let \mathcal{A}_2 be a nondeterministic OPA defined on an OP alphabet (Σ, M) with s states. Then there exists a nondeterministic OPA \mathcal{A}_1 with the same precedence matrix as \mathcal{A}_2 and $O(|\Sigma|^2 s)$ states such that $L(\mathcal{A}_1) = \tilde{L}(\mathcal{A}_2)$.

Proof. Let $\mathcal{A}_2 = \langle \Sigma, M, Q, I, F, \delta \rangle$ and consider, first, an equivalent form for the automaton \mathcal{A}_2 , where all the states are simply enriched with a lookahead and lookback symbol: $\tilde{\mathcal{A}}_2 = \langle \Sigma, M, Q_2, I_2, F_2, \delta_2 \rangle$ where

- $Q_2 = \hat{\Sigma} \times Q \times \hat{\Sigma}$, where $\hat{\Sigma} = (\Sigma \cup \{\#\})$, i.e. the first component of a state is the lookback symbol, the second component of the triple is a state of \mathcal{A}_2 and the third component of the state is the lookahead symbol,
- $I_2 = \{\#\} \times I \times \{a \in \hat{\Sigma} \mid M_{\#a} \neq \emptyset\}$ is the set of initial states of $\tilde{\mathcal{A}}_2$,
- $F_2 = (\{\#\} \cup \{b \in \Sigma : b \succ \#\}) \times F \times \{\#\}$
- and the transition function $\delta_2 : Q_2 \times (\Sigma \cup Q_2) \rightarrow 2^{Q_2}$ is defined in the following natural way
 - $\delta_{2\text{push}}(\langle a, q, b \rangle, b) = \{\langle b, p, c \rangle \mid p \in \delta_{\text{push}}(q, b) \wedge M_{ab} \in \{<, \doteq\} \wedge M_{bc} \neq \emptyset\},$
 $\forall a \in \hat{\Sigma}, b \in \Sigma, q \in Q$
 - $\delta_{2\text{flush}}(\langle a_1, q_1, a_2 \rangle, \langle b_1, q_2, b_2 \rangle) = \{\langle b_1, q_3, a_2 \rangle \mid q_3 \in \delta_{\text{flush}}(q_1, q_2) \wedge M_{a_1 a_2} = \succ$
 $\wedge M_{b_1 a_2} \neq \emptyset\},$
 $\forall a_1, a_2, b_2 \in \Sigma, \forall b_1 \in \hat{\Sigma}, \forall q_1, q_2 \in Q.$

It is clear that $\tilde{L}(\mathcal{A}_2) = \tilde{L}(\tilde{\mathcal{A}}_2)$. Furthermore, the final states of $\tilde{\mathcal{A}}_2$ cannot be reached by flush edges: in fact, if there exists a transition $\langle a_1, q_1, a_2 \rangle \xrightarrow{\langle b_1, q_2, b_2 \rangle} \langle a_1, q_3, \# \rangle$ towards a final state $\langle a_1, q_3, \# \rangle$, then the third component of the flushed and of the reached final state must be equal by definition of the transition function, i.e. $\langle a_1, q_1, a_2 \rangle = \langle a_1, q_1, \# \rangle$. But this flush transition cannot be performed by a variant OPA, which stops a computation right before reading the delimiter #, when the parsing of the word ends.

Hence, one may always refer to a variant OPA assuming that in its graph there are no flush moves towards final states.

It is then possible to describe an automaton OPA \mathcal{A}_1 equivalent to the variant OPA \mathcal{A}_2 (or $\tilde{\mathcal{A}}_2$).

$\mathcal{A}_1 = \langle \Sigma, M, Q_1, I_1, F_1, \delta_1 \rangle$ is defined as $\tilde{\mathcal{A}}_2$ but it is enriched with an additional state, which is the only final state of \mathcal{A}_1 and which is reachable through a flush edge by all final states of $\tilde{\mathcal{A}}_2$. Basically, its role is to let \mathcal{A}_1 empty the stack after parsing a word that is accepted by $\tilde{\mathcal{A}}_2$.

- $Q_1 = Q_2 \cup \{q_{\text{accept}}\}$
- $I_1 = I_2 \cup \{q_{\text{accept}}\}$ if $I_2 \cap F_2 \neq \emptyset$ or $I_1 = I_2$ otherwise
- $F_1 = \{q_{\text{accept}}\}$
- The transition function δ_1 equals δ_2 on all states in Q_2 ; in addition \mathcal{A}_1 has departing flush edges from the final states in F_2 to q_{accept} and q_{accept} has no outgoing push/mark edge but only self-loops flush edges.

The push transition function $\delta_{1\text{push}} : Q_1 \times \Sigma \rightarrow 2^{Q_1}$ is defined as $\delta_{1\text{push}}(q, c) = \delta_{2\text{push}}(q, c)$, $\forall q \in Q_2, c \in \hat{\Sigma}$, whereas $\delta_{1\text{push}}(q_{\text{accept}}, c)$ leads to an error state for any c .

The flush transition $\delta_{1\text{flush}} : Q_1 \times Q_1 \rightarrow 2^{Q_1}$ is defined by:

$$\delta_{1\text{flush}}(q, p) = \delta_{2\text{flush}}(q, p), \forall q, p \in Q_2$$

$$\delta_{1\text{flush}}(q, p) = q_{\text{accept}}, \forall q \in (F_2 \cup \{q_{\text{accept}}\}), p \in Q_2$$

The two automata recognize the same language, $L(\mathcal{A}_1) = \tilde{L}(\tilde{\mathcal{A}}_2)$.

First of all, $L(\mathcal{A}_1) \subseteq \tilde{L}(\tilde{\mathcal{A}}_2)$: in fact, if the OPA \mathcal{A}_1 recognizes a word, then it is either the empty word and thus $q_{\text{accept}} \in I_1$ and also $\tilde{\mathcal{A}}_2$ has a successful run on it, or \mathcal{A}_1 recognizes a word $w \neq \varepsilon$ and there exists a run S of \mathcal{A}_1 which ends in the final state q_{accept} , emptying the stack. Notice that q_{accept} is reached by a flush move from a state in F_2 , say $q_f \in F_2$:

$$S : q_0 \in I_2 \xrightarrow{w} q_f \Longrightarrow q_{\text{accept}} \xrightarrow{p \in Q_1} q_{\text{accept}}^*$$

and q_f itself is reached exactly when the parsing of the word w is finished, since, as said before, a state in F_2 cannot be reached by flush moves. This condition is necessary to avoid the presence of sequences of flush moves from non accepting states towards final states. Then the path from q_0 to q_f , which follows the same state and edges as S , represents a run of $\tilde{\mathcal{A}}_2$ which ends in a final state q_f right after the parsing of the whole word, thus accepting w . The direction from right to left $L(\mathcal{A}_1) \supseteq \tilde{L}(\tilde{\mathcal{A}}_2)$ derives easily from the fact that, if $\tilde{\mathcal{A}}_2$ accepts a word along a successful run, then \mathcal{A}_1 recognizes the word along the same run, possibly emptying the stack in the final state q_{accept} . \square

Given the variant for OPAs on finite words, it is possible to prove the closure under concatenation of the class of languages accepted by ω OPBAs with a language of finite

words accepted by an OPA, as the following theorem (Theorem 2) states. Notice that its proof differs from the non-trivial proof of closure under concatenation of OPLs of finite-length words [6], which, instead, can be recognized deterministically.

Theorem 2. *Let $L_1 \subseteq \Sigma^*$ be a language of finite words recognized by an OPA with OPM M_1 and s_1 states. Let $L_2 \subseteq \Sigma^\omega$ be an ω -language recognized by a nondeterministic ω OPBA with OPM M_2 compatible with M_1 and s_2 states.*

Then the concatenation $L_1 \cdot L_2$ is also recognized by a ω OPBA with OPM $M_3 \supseteq M_1 \cup M_2$ and $O(|\Sigma|^2 s_1^2 + |\Sigma| s_2^2)$ states.

Proof. Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, I_1, F_1, \delta_1 \rangle$ be a nondeterministic OPA which recognizes language L_1 and let $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, I_2, F_2, \delta_2 \rangle$ be a nondeterministic ω OPBA with OPM M_2 compatible with M_1 which accepts L_2 . Suppose, without loss of generality, that Q_1 and Q_2 are disjoint.

To define an automaton ω OPBA \mathcal{A}_3 which accepts the language $L_1 \cdot L_2$, we first build an automaton OPA in the variant form $\mathcal{A}'_1 = \langle \Sigma, M_1, Q'_1, I'_1, F'_1, \delta'_1 \rangle$ such that $\tilde{L}(\mathcal{A}'_1) = L(\mathcal{A}_1)$.

The automaton \mathcal{A}_3 may recognize the first finite words in the concatenation $L_1 \cdot L_2$ simulating \mathcal{A}'_1 : during the parsing of the input string, if \mathcal{A}'_1 reaches a final state at the end of a finite-length prefix, then it belongs to L_1 and \mathcal{A}_3 may immediately start the recognition of the second infinite string without the need to perform any flush move to empty the stack. From this point onwards, then, \mathcal{A}_3 may check that the remaining infinite portion of the input belongs to L_2 , behaving as the ω OPBA \mathcal{A}_2 . Notice, however, that as it happens for operator precedence languages of finite-length words [6], the strings of the concatenation of two OPLs may have syntax trees that significantly differ from the concatenation of the trees of the single words: the trees of the strings of the two languages may be merged, according to the precedence relations between the symbols of the words, in a completely new structure. From the point of view of the parsing of a string in $L_1 \cdot L_2$ by an automaton, the joining of the trees of two words in L_1 and L_2 may imply that the recognition and reduction by flush moves of a subtree with branches in a word in L_1 have to be postponed until the parsing of the other branches in the word in L_2 has been completed. Therefore, \mathcal{A}_3 cannot merely read the second infinite word performing the same transitions as \mathcal{A}_2 , but it is still possible to simulate this ω OPBA keeping in the states some summary information about its runs. In this way, while reading the second word in the concatenation, whenever \mathcal{A}_3 has to reduce a subtree which extends to the previous word in L_1 and thus it has to perform a flush move that involves the portion of the stack piled up during the parsing of the first word, it can still restore on the stack the state that \mathcal{A}_2 would instead have reached, resuming the parsing of the second word thereon as in a run of \mathcal{A}_2 .

In particular, the automaton \mathcal{A}_3 is defined as follows. Let $\hat{\Sigma} = \Sigma \cup \{\#\}$ and $\mathcal{A}_3 = \langle \Sigma, M_3, Q_3, I_3, F_3, \delta_3 \rangle$ where:

- $M_3 \supseteq M_1 \cup M_2$ and may be supposed to be a total matrix, for instance assigning arbitrary precedence relations to the empty entries, so that the concatenation of the languages L_1 and L_2 is well defined.
- $Q_3 = Q'_1 \cup \hat{\Sigma} \times Q_2 \times (Q_2 \cup \{-\})$, i.e. the set of states of \mathcal{A}_3 includes the states of \mathcal{A}'_1 , while the states of \mathcal{A}_2 are extended with two components. The first component

is a lookback symbol, the second component is a state of Q_2 and the third represents, as in the construction for deterministic OPAs [9]), the state with the marked symbol that, when the current input letter is read in a run performed by \mathcal{A}_2 on the infinite substring, is the last marked symbol on the stack. Storing this component is necessary to guarantee that, whenever the automaton \mathcal{A}_3 has to perform a flush move towards states piled in the stack during the recognition of the first word in the concatenation, it is still possible to compute the state that \mathcal{A}_2 would have reached instead.

This third component is denoted ' $-$ ' if all the preceding symbols in the stack have been piled during the parsing of the first word of the concatenation (thus the stack of \mathcal{A}_2 is empty).

- $I_3 = I'_1 \cup \{\langle \#, p_0, - \rangle \mid p_0 \in I_2\}$ if $\varepsilon \in L_1$ or $I_3 = I'_1$ otherwise
- $F_3 = \hat{\Sigma} \times F_2 \times Q_2$
- The transition function $\delta_3 : Q_3 \times (\Sigma \cup Q_3) \rightarrow 2^{Q_3}$ is defined as follows. The push transition $\delta_{3\text{push}} : Q_3 \times \Sigma \rightarrow 2^{Q_3}$ is defined by:
 - $\delta_{3\text{push}}(q_1, c) = \delta'_{1\text{push}}(q_1, c), \forall q_1 \in Q'_1, c \in \Sigma$, i.e. it simulates \mathcal{A}'_1 on Q'_1
 - $\delta_{3\text{push}}(q_1, c) = \{\langle \#, p_0, - \rangle \mid p_0 \in I_2\}, \forall q_1 \in Q'_1, c \in \Sigma : \exists q_f \in F'_1$ s.t.
 $\delta'_{1\text{push}}(q_1, c) \ni q_f$,
 i.e. it reaches the initial states of \mathcal{A}_2 after the recognition of a word in L_1
 - $\delta_{3\text{push}}(\langle a, p, r \rangle, c) = \begin{cases} \{\langle c, q, p \rangle \mid q \in \delta_{2\text{push}}(p, c)\} & \text{if } a \prec c \\ \{\langle c, q, r \rangle \mid q \in \delta_{2\text{push}}(p, c)\} & \text{if } a \doteq c \end{cases}$
 for $a \in \hat{\Sigma}, c \in \Sigma, p \in Q_2, r \in (Q_2 \cup \{-\})$

The flush transition $\delta_{3\text{flush}} : Q_3 \times Q_3 \rightarrow 2^{Q_3}$ is defined by:

- $\delta_{3\text{flush}}(q_1, p_1) = \delta'_{1\text{flush}}(q_1, p_1), \forall q_1, p_1 \in Q'_1$, i.e. it simulates \mathcal{A}'_1 on Q'_1
- $\delta_{3\text{flush}}(\langle \#, p, - \rangle, q) = \langle \#, p, - \rangle$, with $p \in Q_2, q \in Q'_1$
- $\delta_{3\text{flush}}(\langle a_1, p_1, r_1 = p_2 \rangle, \langle a_2, p_2, r_2 \rangle) = \{\langle a_2, q, r_2 \rangle \mid q \in \delta_{2\text{flush}}(p_1, p_2)\}$,
 where $a_1 \in \Sigma, a_2 \in \hat{\Sigma}$
- $\delta_{3\text{flush}}(\langle a, p, r \rangle, q) = \{\langle \#, s, - \rangle \mid s \in \delta_{2\text{flush}}(p, r)\}$, for $a \in \Sigma, p, r \in Q_2, q \in Q'_1$
 i.e. whenever the precedence relations induce a merging of the subtrees of the words of the concatenation, \mathcal{A}_3 restores the state s at the bottom of the stack of \mathcal{A}_2 from which a run of \mathcal{A}_2 will continue.

It is clear that the ω OPBA \mathcal{A}_3 recognizes $L_1 \cdot L_2$, thus the class of languages accepted by ω OPBA is closed under concatenation on the left with languages recognized by OPAs. \square

Closure under complementation

Theorem 3. *Let M be a conflict-free precedence matrix on an alphabet Σ . Denote by $L_M \subseteq \Sigma^\omega$ the ω -language comprising all infinite words $x \in \Sigma^\omega$ compatible with M . Let L be an ω -language on Σ that can be recognized by a nondeterministic ω OPBA with precedence matrix M and s states. Then the complement of L w.r.t L_M is recognized by an ω OPBA with the same precedence matrix M and $2^{O(s^2)}$ states.*

Proof. The proof follows to some extent the structure of the corresponding proof for Büchi VPAs [1], but it exhibits some relevant technical aspects which distinctly characterize it; in particular, we need to introduce an ad-hoc factorization of ω -words due to the more complex management of the stack performed by ω OPAs.

Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be a nondeterministic ω OPBA with $|Q| = s$. Without loss of generality \mathcal{A} can be considered complete with respect to the transition function δ , i.e. there is a run of \mathcal{A} on every ω -word on Σ compatible with M .

In general, a sentence on Σ^ω can be factored in a unique way so as to distinguish the subfactors of the string that can be recognized without resorting to the stack of the automaton and those subwords for which the use of the stack is necessary.

More precisely, an ω -word $w \in \Sigma^\omega$ can be factored as a sequence of chains and pending letters $w = w_1 w_2 w_3 \dots$ where either $w_i = a_i \in \Sigma$ is a pending letter or $w_i = a_{i1} a_{i2} \dots a_{in}$ is a finite sequence of letters such that $\langle l_i w_i^{first_{i+1}} \rangle$ is a chain, where l_i denotes the last pending letter preceding w_i in the word and $first_{i+1}$ denotes the first letter of word w_{i+1} . Let also, by convention, $a_0 = \#$ be the first pending letter.

Notice that such factorization is not unique, since a string w_i can be nested into a larger chain having the same preceding pending letter. The factorization is unique, however, if we additionally require that w_i has no prefix which is a chain.

As an example, for the word $w = \langle a \rangle \langle c \rangle b \langle a \rangle \langle d \rangle b \dots$, with precedence relations in the OPM $a \succ b$ and $b \prec d$, the unique factorization is $w = w_1 b w_3 w_4 b \dots$, where b is a pending letter and $\langle^{\#} a c^b \rangle, \langle^b a d \rangle, \langle^b d^b \rangle$ are chains.

Define a *semisupport for the simple chain* $\langle^{a_0} a_1 a_2 \dots a_n^{a_{n+1}} \rangle$ as any path in \mathcal{A} of the form

$$q_0 \xrightarrow{a_1} q_1 \longrightarrow \dots \longrightarrow q_{n-1} \xrightarrow{a_n} q_n \xRightarrow{q_0} q_{n+1} \quad (5)$$

A *semisupport for the composed chain*, with no prefix that is a chain, $\langle^{a_0} a_1 x_1 a_2 \dots a_n x_n^{a_{n+1}} \rangle$ is any path in \mathcal{A} of the form

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{x_1} q'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \xrightarrow{x_n} q'_n \xRightarrow{q_0} q_{n+1} \quad (6)$$

where, for every $i : 1 \leq i \leq n$:

- if $x_i \neq \varepsilon$, then $\xrightarrow{a_i} q_i \xrightarrow{x_i} q'_i$ is a support for the chain $\langle^{a_i} x_i^{a_{i+1}} \rangle$, i.e., it can be decomposed as $\xrightarrow{a_i} q_i \xrightarrow{x_i} q''_i \xRightarrow{q_i} q'_i$.
- if $x_i = \varepsilon$, then $q'_i = q_i$.

Unlike the definition of the support for a simple (Equation 1) and a composed chain (Equation 2), in a semisupport for a chain the initial state q_0 is not restricted to be the state reached after reading symbol a_0 .

Let $x \in \Sigma^*$ be such that $\langle^a x^b \rangle$ is a chain for some a, b and let $T(x)$ be the set of all triples $(q, p, f) \in Q \times Q \times \{0, 1\}$ such that there exists a semisupport $q \xrightarrow{x} p$ in \mathcal{A} , and $f = 1$ iff the semisupport contains a state in F . Also let \mathcal{T} be the set of all such $T(x)$, i.e., \mathcal{T} contains set of triples identifying all semisupports for some chain, and set $PR = \Sigma \cup \mathcal{T}$. The *pseudorun* for w in \mathcal{A} is the ω -word $w' = y_1 y_2 y_3 \dots \in PR^\omega$ where $y_i = a_i$ if $w_i = a_i$, otherwise $y_i = T(w_i)$.

For the example above, then, $w' = T(ac) b T(a) T(d) b \dots$

We now define a nondeterministic Büchi finite-state automaton \mathcal{A}_R over alphabet PR that recognizes the pseudorun w' of any $w \in L(\mathcal{A})$. \mathcal{A}_R has all states of \mathcal{A} and transitions corresponding to \mathcal{A} 's push transitions but it is devoid of flush edges (indeed they cannot be taken by a regular automaton without a stack). In addition, for every $S \in \mathcal{T}$ it is endowed with arcs labeled S which link, for each triple (q, p, f) in S , either the pair of states q, p or q, p' if $f = 1$, where p' is a new final state which summarizes the states in F met along the semisupport $q \rightsquigarrow p$ and which has the same outgoing edges as p .

Notice that, given a set $S \in \mathcal{T}$, the existence of an edge S between the pairs of states q, p in the triples in S can be decided in an effective way.

The automaton \mathcal{A}_R built so far is able to parse all pseudoruns and recognizes all pseudoruns of ω -words recognized by \mathcal{A} . However, since its moves are no longer determined by the OPM M , it can also accept input words along the edges of the graph of \mathcal{A} which are not pseudorun since they do not correspond to a correct factorization on PR . This is irrelevant, however, since the aim of the proof is to devise an automaton recognizing the complement of $L(\mathcal{A})$, and all the words in $L_M \setminus L(\mathcal{A})$ are parsed along pseudoruns, which are not accepted by \mathcal{A}_R . If one gives as input words only pseudoruns (and not generic words on PR), then they will be accepted by \mathcal{A}_R if the corresponding words on Σ belong to $L(\mathcal{A})$, and they will be rejected if the corresponding words do not belong to $L(\mathcal{A})$. Given the Büchi finite-state automaton \mathcal{A}_R (which has $O(s)$ states), one can now construct a deterministic Streett automaton \mathcal{B}_R that accepts the complement of $L(\mathcal{A}_R)$, on the alphabet PR . If \mathcal{B}_R receives as input words on PR only pseudoruns, then it will accept only words in $L_M \setminus L(\mathcal{A})$. The automaton \mathcal{B}_R has $2^{O(s \log s)}$ states and $O(s)$ accepting constraints [16].

Consider then a nondeterministic transducer ω OPBA \mathcal{B} that on reading w generates online the aforementioned pseudorun w' , which will be given as input to \mathcal{B}_R . The automaton \mathcal{B} nondeterministically guesses whether the next input symbol is a pending letter, the beginning of a chain appearing in the factorization of w , or a symbol within such a chain, and uses stack symbols Z, \perp , or elements in \mathcal{T} , respectively, to distinguish these three cases.

In order to produce w' , whenever the automaton reads a pending letter it outputs the letter itself, whereas when it ends to recognize a chain of the factorization, performing a flush move towards a state with \perp as first component, it outputs the set of all the pairs of states which define a semisupport for the chain. Thus, the output w' produced by \mathcal{B} is unique, despite the nondeterminism of the translator.

Formally, the transducer ω OPBA $\mathcal{B} = \langle \Sigma, M, Q_B, I_B, F_B, PR, \delta_B, \eta_B \rangle$ is defined as follows:

- $Q_B = \hat{\Sigma} \times (\{Z, \perp\} \cup \mathcal{T})$ where $\hat{\Sigma} = \Sigma \cup \{\#\}$. The first component of a state in Q_B denotes the lookback symbol read to reach the state, the second component represents the guess whether the next symbol to be read is a pending letter (Z), the beginning of a chain (\perp), or a letter within such a chain w_i ($T \in \mathcal{T}$). In the third case, T contains all information necessary to correctly simulate the moves of \mathcal{A} during the parsing of the chain w_i of w , and compute the corresponding symbol y_i of w' . In particular, T is a set comprising all triples (r, q, v) where r represents the state reached before the last mark move, q represents the current state reached by

\mathcal{A} , and ν is a bit that reminds whether, while reading the chain, a state in F has been encountered (as in the construction of a deterministic OPA on words of finite length [9], it is necessary to keep track of the state from which the parsing of a chain started, to avoid erroneous merges of runs on flush moves).

- $I_B = \{\langle \#, \perp \rangle, \langle \#, Z \rangle\}$.
- $F_B = \{\langle a, \perp \rangle, \langle a, Z \rangle \mid a \in \hat{\Sigma}\}$.
- The transition function and the output function are defined as the union of two disjoint pairs of functions. Let $a \in \hat{\Sigma}$, $b, c \in \Sigma$, $T, S \in \mathcal{T}$. The push pair $\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle : Q_B \times \Sigma \rightarrow \mathcal{P}_F(Q_B \times PR^*)$ is defined as follows, where the symbols after \downarrow denotes the output of the move of the automaton.
 - *Push of a pending letter.*

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle(\langle a, Z \rangle, b) = \{\langle b, \perp \rangle \downarrow b, \langle b, Z \rangle \downarrow b\}$$

- *Mark at the beginning of a chain of the factorization.* If $a \prec b$ then:

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle(\langle a, \perp \rangle, b) = \{\langle b, T \rangle \downarrow \varepsilon\}$$

$$\text{where } T = \{\langle q, p, \nu \rangle \mid q \in Q, p \in \delta_{\text{push}}(q, b), \nu = 1 \text{ iff } p \in F\}$$

- *Push within a chain of the factorization.*

$$\langle \delta_{\text{Bpush}}, \eta_{\text{Bpush}} \rangle(\langle a, T \rangle, b) = \{\langle b, S \rangle \downarrow \varepsilon\} \quad \text{where}$$

$$S = \left\{ \langle t, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T \text{ s.t. } t = \begin{cases} q & \text{if } a \prec b \\ r & \text{if } a \doteq b \end{cases}, \nu = \begin{cases} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{cases}, p \in \delta_{\text{push}}(q, b) \right\}$$

The flush pair $\langle \delta_{\text{Bflush}}, \eta_{\text{Bflush}} \rangle : Q_B \times Q_B \rightarrow \mathcal{P}_F(Q_B \times PR^*)$ is defined as follows.

- *Flush at the end of a chain of the factorization.*

$$\langle \delta_{\text{Bflush}}, \eta_{\text{Bflush}} \rangle(\langle b, T \rangle, \langle a, \perp \rangle) = \{\langle a, \perp \rangle \downarrow R, \langle a, Z \rangle \downarrow R\} \quad \text{where}$$

$$R = \left\{ \langle r, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T, \text{ s.t. } p \in \delta_{\text{flush}}(q, r), \nu = \begin{cases} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{cases} \right\}$$

- *Flush within a chain of the factorization.*

$$\langle \delta_{\text{Bflush}}, \eta_{\text{Bflush}} \rangle(\langle b, T \rangle, \langle c, S \rangle) = \{\langle c, R \rangle \downarrow \varepsilon\} \quad \text{where}$$

$$R = \left\{ \langle t, p, \nu \rangle \mid \exists \langle r, q, \xi \rangle \in T, \exists \langle t, r, \zeta \rangle \in S \text{ s.t. } p \in \delta_{\text{flush}}(q, r), \nu = \begin{cases} \xi & \text{if } p \notin F \\ 1 & \text{if } p \in F \end{cases} \right\}$$

An error state is reached for any other case. In particular, no flush move is defined when the second state has Z as second component, nor when the first state has Z or \perp as second component, as consistent with the meaning of stack symbol Z and \perp .

In the end, the final automaton to be built, which recognizes the complement of $L = L(\mathcal{A})$ w.r.t L_M , is the ω OPBA representing the product of \mathcal{B}_R (converted to a Büchi automaton), which has $2^{O(s \log s)}$ states, and \mathcal{B} , which has $|Q_B| = 2^{O(s^2)}$ states: while reading w , \mathcal{B} outputs the pseudorun w' of w online, and the states of \mathcal{B}_R are updated accordingly. The automaton accepts if both \mathcal{B} and \mathcal{B}_R reach infinitely often final states. Furthermore, it has $2^{O(s^2)}$ states. \square

4.1 Closure properties of $\mathcal{L}(\omega\text{DOPBA})$ under intersection and union

The class of languages accepted by ωDOPBA s is closed under intersection and union.

Closure under intersection

Theorem 4. *Let L_1 and L_2 be ω -languages that can be recognized by two ωDOPBA s defined over the same alphabet Σ , with compatible precedence matrices M_1 and M_2 and s_1 and s_2 states respectively. Then $L = L_1 \cap L_2$ is recognizable by a ωDOPBA with OPM $M = M_1 \cap M_2$ and $O(s_1 s_2)$ states.*

Proof. The proof derives from the analogous proof of closure with respect to intersection of languages recognized by ωOPBA s described in [13]. In fact the ωOPBA which accepts the intersection of two languages L_1 and L_2 recognized by two ωOPBA s \mathcal{A}_1 and \mathcal{A}_2 with compatible OPMs described in that proof is deterministic if both the automata \mathcal{A}_1 and \mathcal{A}_2 are deterministic. \square

Closure under union

Theorem 5. *Let L_1 and L_2 be ω -languages that can be recognized by two ωDOPBA s defined over the same alphabet Σ , with compatible precedence matrices M_1 and M_2 and s_1 and s_2 states respectively. Then $L = L_1 \cup L_2$ is recognizable by an ωDOPBA with OPM $M = M_1 \cup M_2$ and $O(s_1 s_2)$ states.*

Proof. Let $\tilde{\mathcal{A}}_1 = \langle \Sigma, M_1, \tilde{Q}_1, \tilde{q}_{01}, \tilde{F}_1, \tilde{\delta}_1 \rangle$ and $\tilde{\mathcal{A}}_2 = \langle \Sigma, M_2, \tilde{Q}_2, \tilde{q}_{02}, \tilde{F}_2, \tilde{\delta}_2 \rangle$ be two ωDOPBA s accepting the languages $L(\tilde{\mathcal{A}}_1) = L_1$ and $L(\tilde{\mathcal{A}}_2) = L_2$ and with compatible precedence matrices M_1 and M_2 . Suppose without loss of generality that \tilde{Q}_1 and \tilde{Q}_2 are disjoint. Let $|\tilde{Q}_1| = s_1$ and $|\tilde{Q}_2| = s_2$.

Since M_1 and M_2 are compatible, then $M = M_1 \cup M_2$ is conflict-free and the two ωDOPBA s may be normalized completing their precedence matrix to $M = M_1 \cup M_2$ (see e.g. the normalization described in [13]). The normalization preserves the determinism of the automata and keeps their sets of states disjoint.

The automata may be, then, completed as regards their transition function, so that there is a run on their graph for every ω -word in L_M [13]. The completed automata $\mathcal{A}_1 = \langle \Sigma, M = M_1 \cup M_2, Q_1, q_{01}, F_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M = M_1 \cup M_2, Q_2, q_{02}, F_2, \delta_2 \rangle$ are still deterministic with disjoint state sets and recognize the same languages as $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$, i.e. $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$. Furthermore, $|Q_1| = O(s_1)$ and $|Q_2| = O(s_2)$.

An ωDOPBA \mathcal{A}_3 which recognizes $L_1 \cup L_2$ may then be defined adopting the usual product construction for regular automata: $\mathcal{A}_3 = \langle \Sigma, M = M_1 \cup M_2, Q_3, q_{03}, F_3, \delta_3 \rangle$ where:

- $Q_3 = Q_1 \times Q_2$,
- $q_{03} = (q_{01}, q_{02})$,
- $F_3 = F_1 \times Q_2 \cup Q_1 \times F_2$

- and the transition function $\delta_3 : Q_3 \times (\Sigma \cup Q_3) \rightarrow Q_3$ is defined as follows. The push transition $\delta_{3\text{push}} : Q_3 \times \Sigma \rightarrow Q_3$ is expressed as:

$$\delta_{3\text{push}}((q_1, q_2), a) = (\delta_{1\text{push}}(q_1, a), \delta_{2\text{push}}(q_2, a))$$

$$\forall q_1 \in Q_1, q_2 \in Q_2, a \in \Sigma.$$

The flush transition $\delta_{3\text{flush}} : Q_3 \times Q_3 \rightarrow Q_3$ is defined as:

$$\delta_{3\text{flush}}((q_1, q_2), (p_1, p_2)) = (\delta_{1\text{flush}}(q_1, p_1), \delta_{2\text{flush}}(q_2, p_2))$$

$$\forall q_1, p_1 \in Q_1, q_2, p_2 \in Q_2$$

The $\omega\text{DOPBA } \mathcal{A}_3$ simulates \mathcal{A}_1 and \mathcal{A}_2 respectively on the two components of the states, and accepts an ω -word iff there is an accepting run on it for at least one of the two automata.

The definition of the transition function is sound because the automata \mathcal{A}_1 and \mathcal{A}_2 have the same precedence matrix, thus they perform the same type of move (mark/push/flush) while reading the input word; furthermore, they are both complete w.r.t their transition function and none of them may stop a computation while reading a string. \square

5 Conclusions and further research

We presented a formalism for infinite-state model checking based on operator precedence languages, continuing to explore the paths in the lode of operator precedence languages started up by Robert Floyd a long time ago. We introduced various classes of automata able to recognize operator precedence languages of infinite-length words whose expressive power outperforms classical models for infinite-state systems as Visibly Pushdown ω -languages, allowing to represent more complex systems in several practical contexts. We proved the closure properties of ωOPLs under Boolean operations that, along with the decidability of the emptiness problem, are fundamental for the application of such a formalism to model checking. For instance, with reference to Example 2, imagine that one builds a specialized system that includes only procedures of type a and where interrupts of lowest level are disabled when there is any pending call_a : once having built a new model $\hat{\mathcal{A}}$ for such a system she can automatically verify its compliance with the more general one \mathcal{A} by checking whether $L(\hat{\mathcal{A}}) \subseteq L(\mathcal{A})$.

Our results open further directions of research. A first topic deals with the investigation of properties and fields of application of OPAs and ωOPAs as transducers, as they may e.g. translate tagged documents written in mark-up languages (as XML, HTML) into the final displayed (XML, HTML) page, or they may translate the traces of operations of do-undo actions performed on different versions of a file into an end-user log or document. Thus, it might be possible to define a formal translation from structured or semistructured languages or patterns of tasks and client behaviors into suitable final-user views of the model.

A second interesting research issue is the characterization of ωOPLs in terms of suitable monadic second order logical formulas, that has already been studied for operator precedence languages of finite-length strings [11]. This would further strengthen applicability of model checking techniques. The next step of investigation will regard the actual design and study of complexity issues of algorithms for model checking of expressive logics on these pushdown models. We expect that the peculiar features of

operator precedence languages, as their “locality principle” which makes them suitable for parallel and incremental parsing [2,3] and their expressivity, might be interestingly exploited to devise efficient and attractive software model-checking procedures and approaches.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. *Journ. ACM* 56(3) (2009)
2. Barengi, A., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: Parallel parsing of operator precedence grammars. *Information Processing Letters* (2013), to appear
3. Barengi, A., Viviani, E., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: PAPAGENO: a parallel parser generator for operator precedence grammars. In: 5th International Conference on Software Language Engineering (SLE) (2012)
4. Büchi, J.R.: Weak Second-Order Arithmetic and Finite Automata. *Mathematical Logic Quarterly* 6(1-6), 66–92 (1960)
5. Burkart, O., Steffen, B.: Model checking for context-free processes. In: *CONCUR '92*, LNCS, vol. 630, pp. 123–137 (1992)
6. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. *Journal of Computer and System Science* 78(6), 1837–1867 (2012)
7. Floyd, R.W.: Syntactic Analysis and Operator Precedence. *Journ. ACM* 10(3), 316–333 (1963)
8. Grune, D., Jacobs, C.J.: *Parsing techniques: a practical guide*. Springer, New York (2008)
9. Lonati, V., Mandrioli, D., Pradella, M.: Precedence Automata and Languages. *CoRR abs/1012.2321* (2010)
10. Lonati, V., Mandrioli, D., Pradella, M.: Precedence Automata and Languages. In: *The 6th International Computer Science Symposium in Russia (CSR)*, LNCS, vol. 6651, pp. 291–304 (2011)
11. Lonati, V., Mandrioli, D., Pradella, M.: Logic Characterization of Invisibly Structured Languages: the Case of Floyd Languages. *SOFSEM* (to appear) (2013)
12. Muller, D.E.: Infinite sequences and finite machines. In: *Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*. pp. 3–16. SWCT '63, IEEE Computer Society, Washington, DC, USA (1963)
13. Panella, F.: Floyd languages for infinite words. Master's thesis, Politecnico di Milano (2011), <http://home.dei.polimi.it/panella>
14. Rabin, M.: Automata on infinite objects and Church's problem. *Regional conference series in mathematics*, Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society (1972)
15. Streett, R.S.: Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control* 54(1-2), 121 – 141 (1982)
16. Thomas, W.: *Handbook of theoretical computer science* (vol. B). chap. Automata on infinite objects, pp. 133–191. MIT Press, Cambridge, MA, USA (1990)